43466

National Library of Canada
Canadian Theses Division
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Division des thèses canadiennes

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

Robert WAYNE LEWIS

Date of Birth — Date de naissance
~~18~~ 03/31/48

Country of Birth — Lieu de naissance
CANADA

Permanent Address — Résidence fixe
#126 HILLCREST PLACE
159 St & 77A Ave T5R 5X6
EDMONTON, ALTA

Title of Thesis — Titre de la thèse
A MICROPROCESSOR CONTROLED
BRAILLE TRANSCRIBER

University — Université
U. OF A

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée
M. Sc

Year this degree conferred — Année d'obtention de ce grade
1979

Name of Supervisor — Nom du directeur de thèse
PAT HARDING

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date
~~Sept~~ 24/79

Signature
R. W. Lewis

NL-91 (4/77)

National Library of Canada

Cataloguing Branch
Canadian Theses Division

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada

Direction du catalogage
Division des thèses canadiennes

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

THE UNIVERSITY OF ALBERTA

A MICROPROCESSOR CONTROLLED BRAILLE TRANSCRIPTION SYSTEM

by

ROBERT W. LEWIS

(C)

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

IN DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL, 1979

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled A Microprocessor Controlled Braille Transcription System submitted by Bob Lewis in partial fulfilment of the requirements for the degree of Master of Science.

D. A. Hamburger

Supervisor

Keith Stevenson

Date Sept 14 1979

Abstract

This thesis details the construction of both the software and hardware of a microprocessor controlled Braille transcriber. The Braille transcriber is operated by a sighted typist and transcribes English into Braille at the grade 2 level of Braille. Special skills are not needed to operate this system. The Braille output is embossed on paper by a Braille printer. The printer was designed specifically for this system and is handled by the microprocessor as a special output device.

Acknowledgements

The author wishes to express his appreciation to his supervisor Pat Harding for technical assistance during the course of this work and during the preparation of this thesis. The assistance of Doris Gates of the Canadian National Institute of the Blind in providing assistance with the rules of Braille, is noted with appreciation.

The assistance of the Electrical Engineering machine shop is also gratefully acknowledged in the design and building of the Braille printer, in particular George Fiz.

Financial assistance for this project was provided by professors , Dr. J. Kingma and Mr. P. Harding and is gratefully acknowledged.

The author would like to extend sincere thanks to his wife for her constant encouragement through this work.

TABLE OF CONTENTS

CHAPTER PAGE

I. INTRODUCTION.....1

II. SYSTEM OPERATION.....14

TRANSCRIPTION CONTROL COMMAND.....17

III. SYSTEM CONFIGURATION.....18

A. SYSTEM HARDWARE.....18

ELECTRONICS.....20

MECHANICAL.....43

B. SOFTWARE STRUCTURE.....45

INITIALIZATION.....49

INTERUPT DECODING.....51

CONTROL.....52

TEXT EDITOR.....53

TRANSCRIPTION ROUTINES.....55

EXECUTIVE ROUTINE.....60

PUNCTUATION ROUTINE.....61

SEARCH ROUTINE.....61

I/O ROUTINES.....62

IV. BRAILLE MACRO ROUTINES.....67

A. SIMPLE UPPER WORDSIGNS.....69

B. WORDSIGNS.....69

C. CONTRACTIONS.....70

D. UPPER CONTRACTIONS.....70

E. LOWER CONTRACTIONS.....72

F. LOWER WORDSIGNS.....74

G. INITIAL WORDSIGNS.....77

H. FINAL CONTRACTIONS.....81

V. SUMMARY.....82

VI. REFERENCES.....84

VII. APPENDIX A: THE RULES OF ENGLISH BRAILLE.....86

A. SIMPLE UPPER WORDSIGNS.....86

B. WORDSIGNS.....87

C. CONTRACTIONS.....87

UPPER CONTRACTIONS.....88

LOWER CONTRACTIONS.....89

LOWER WORDSIGNS.....89

COMPOUND SIGNS.....90

INITIAL WORDSIGNS.....91

FINAL CONTRACTIONS.....92

VIII. APPENDIX B: MCS6502 PROGRAMMING ARCHITECTURE.....93

IX. APPENDIX C: EDIT COMMANDS.....94

X. APPENDIX D: PROGRAM MEMORY MAP.....97

XI. APPENDIX E: SOURCE LISTINGS.....101

A. INITIALIZATION RCUTINES.....101

SYSTEM INITIALIZATION.....101

INITIALIZE TEXT BUFFER.....101

INITIALIZE PRINT BUFFER AND VIDEO POINTERS.102

B. INTERUPT DRIVEN ROUTINES.....102

INTERUPT DECODING.....102

CURSOR.....103

C. TEXT EDITOR ROUTINES.....103

CURSOR ALTERNATE ROUTINE.....103
KEYBOARD DECODE.....104
FORWARD SPACE CURSOR.....105
BACK SPACE CURSOR.....105
CLEAR ENTIRE LINE.....106
REVERSE LINE FEED.....106
ADVANCE LINE FEED.....106
HOME CURSOR.....107
CLEAR TO THE END OF LINE.....107
HOME TO BEGINING OF TEXT.....107
CURSOR MOVED RIGHT BY 16.....108
FORWARD LINEFEED BY 8 LINES.....108
ESCAPE DECODING.....108
INSERT.....109
TEXT+1.....109
DELETE TEXT.....110
DELETE CHARACTER AND ADJUST.....110
CLEAR THE ENTIRE PAGE.....111
CLEAR TO THE END OF THE PAGE.....111
CLEAR DISPLAY.....112
CURSOR FEED.....112
D. BRAILLE EXECUTIVE ROUTINES.....113
SYSTEM CONTROL ROUTINE.....113
FILE BLOCK MANAGEMENT.....113
CHANGE TEXT.....114
REPLACE THE TEXT.....114
STORE IN PRINT BUFFER.....115

MAP.....116
FILE TO PRINT BUFFER.....116
WRITE.....116
COPY.....117
RESTART.....117
RESET.....118
TERMINATION CHECK.....118
TERMINATION CHARACTER.....118
E. SEARCHING, SORTING AND CHECKING ROUTINES.....119
CHECK.....119
PUNCTUATION SUCEEDING.....119
DECREMENT POINTER.....119
PREVIOUS CHARACTER.....120
CHECK NEXT CHARACTER.....120
PUNCTUATION PRECEEDING.....120
SORT.....121
CHARACTER.....122
DASH SUCEEDING.....122
DASH PRECEEDING.....122
CHECK NUMBER.....122
SET POINTER.....123
SYLLABLE OVERLAP.....123
SEARCH FOR TWO CHARACTERS.....124
CHECK WHOLE WORD CONTRACTION.....125
TABLE 3.....126
DIPHTHONG OR DIAERESE.....126
DOT ONE AND FOUR CHECK.....126

LOWER CONTRACTION CHECK.....127
LOWER CONTRACTION VOWEL CHECK.....128
SEARCH FOR BE PREFIX.....129
SAVE TEXT.....129
CHANGE THE INDIRECT POINTER.....129
CHECK A AND RETURN CODE.....130
SEARCH TEXT.....130
CHECKA.....132
LAST CHARACTER CHECK.....132
PAST CHARACTER.....133
PREFIX.....133
VOWEL.....134
SUFFIX.....136
PUNCTUATION ROUNTINE.....136
NUMBER ROUTINE.....145
SEARCH ROUTINE.....148
WORDSIGN PARAMETER INITIALIZATION.....151
F. BRAILLE TRANSCRIPTION MACRO ROUTINES.....152
SIMPLE UPPER WORDSIGNS.....152
WHOLE WORD CONTRACTIONS.....152
UPPER CONTRACTIONS.....153
WHOLE WORD CONTRACTIONS.....153
UPPER CONTRACTIONS, "ED, ER, OU, OW".....154
UPPER CONTRACTION,"ST.....154
UPPER CONTRACTION, "AR".....154
UPPER CONTRACTIONS "ING, BLE".....155
LOWER CONTRACTION "COM".....155

LOWER CONTRACTIONS "CON, DIS, BE".....155

LOWER CONTRACTIONS "BB, CC, DD, FF, GG"....156

LOWER CONTRACTION "EN" AND WORDSIGN "ENOUGH"..
156

LOWER WORDSIGN "IN".....157

BE.....157

EA.....157

LOWER WORDSIGNS "INTO, TO, BY".....158

FINAL CONTRACTIONS.....159

COMPOUND SIGNS "EVER, HERE".....160

INITIAL WORDSIGNS "TIME, SOME, PART, ONE"..160

INITIAL WORDSIGNS "UNDER, WORD".....161

XII. REFERENCE TABLES.....162

A. ABBREVIATION TABLES.....162

10.....162

9.....163

8.....164

7.....164

6.....166

5.....167

4.....168

3.....169

B. WORDSIGN TABLE.....169

9.....169

8.....169

7.....169

6.....170

5.....170

4.....172
3.....175
2.....177
C. ALPHABET AND NUMBER TABLES.....180
D. PREFIX TABLES.....181
5.....181
4.....182
3.....183
2.....184
E. SUFFIX TABLES.....186
6.....186
5.....186
4.....187
3.....188
2.....189
F. INDEXED TABLES.....191
PREADD.....191
PRETAB.....191
SUFADD.....192
SUFTAB.....192
WSPNT.....192
WRDTBW.....193
WOFFST.....193
ABVPNT.....194
WDTBAB.....194
ABOFST.....195

LIST OF FIGURES

FIGURE		PAGE
1.	GRADE ONE AND GRADE TWO BRAILLE	3
2.	CONTROL BOARD - SINGLE STEP LOGIC	7
3.	CONTROL BOARD - ADDRESS BREAK POINT LOGIC	8
4.	SYSTEM CONFIGURATION	16
5.	SYSTEM BLOCK DIAGRAM	19
6.	PROGRAMING MODEL	26
7.	CPU BOARD -PROCESSOR AND ADDRESS DECODING	29
8.	CPU BOARD -I/O AND ROM	30
9.	ROM AND CASSETTE BOARD	33
10.	RAM BOARD -BANK 0 AND 1	35
11.	RAM BOARD -BANK 2 AND 3	36
12.	RAM BOARD -ADDRESS DECODING AND DRIVERS	37
13.	VIDEO BOARD -WITH MODIFICATIONS	40
14.	SOLENOID AND MOTOR DRIVER BOARD	42
15.	BRAILLE PRINTER	44
16.	MEMORY MAP	48

I. Introduction

Braille language was developed in 1829 by Louis Braille, for the purpose of making printed material available to the blind. Braille characters are represented by a three by two matrix of dots.

There are two accepted standards of Braille, grade 1 and grade 2. English that is transcribed directly in a one to one mapping of English symbols to Braille symbols is refered to as grade 1 Braille. The grade 1 level of Braille is not widely used because the Braille material would occupy an excessive volume in most instances. The reading rate with grade 1 Braille is also much slower than with grade 2 Braille. Grade 1 Braille is however, useful as a learning aid for new students of the Braille language. The second level of Braille contains many contractions and abbreviations; this feature allows a faster reading rate than with grade 1 Braille and also permits a reduction in the physical volume of the material. The grade 2 level is the accepted standard and is used in almost all commercially available Braille material.

There are many contractions and abbreviations associated with grade 2 Braille. The exceptions to the rules governing the contractions and abbreviations are the main source of difficulty in transcription. Transcribing Braille according to these restrictions requires extensive training

and therefore limits the general availability of Braille. The remainder of this thesis will deal with grade 2 level Braille and all references to Braille refer to grade 2 Braille. An example of grade 2 braille and grade 1 Braille is given in figure 1 (1).

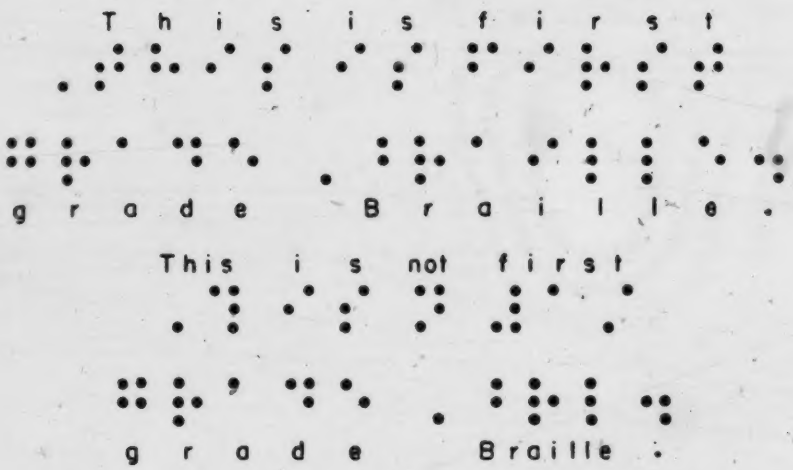

Fig. 1 Grade One and Grade Two Braille

There are two methods of transcription of Braille in use today, human transcripton and computer transcription. The problems associated with human transcription are readily apparent; low volume, slow turn around time, high cost and the necessity to use trained personel. This last requirement is a significant restriction. The lack of skilled transcribers restricts the quantity of material from this source. Human transcription is generally used where commercial material is not readily available.

Computer programs run on large mainframe computers and modern minicomputers have in general made Braille material more readily available to the blind however, time delay, expense and general inaccessability still are prevalent. A machine that would transcribe English to Braille from an English text input, be portable, relatively inexpensive and could be operated by any sighted or unsighted typist would be a major advancement in helping the blind to be more independant and self sufficient. This machine would prove to be of benift as well, to the sighted person wanting to communicate in writing to the blind.

The purpose of this project was to construct and program a microprocessor computer system that would transcribe English to Braille. The machine is to produce Braille at the grade 2 level with an acceptable error rate. The errors produced will be due to contraction and abbreviation rules being contravened and this error rate will be small in standard English text. For example the

syllable 'be' in bestrew is an allowable Braille contraction however, the algorithm for this contraction will not allow this contraction and the syllable 'be' is left uncontracted resulting in the use of more space. The word is transcripted without error however.

The transcriber should have the capability of being designed into a compact portable machine, and with modern production methods should have a low final cost. This thesis describes such a machine and how it has been realized. This machine will be referred to as the 'transcriber' throughout the rest of the thesis.

In the construction of the transcriber, a system using descrete logic design was not given serious consideration since a large number of complicated decisions are needed to implement the transcription algorithms. The algorithms of transcription can most effectively be executed by a computer with an adequate instruction set in conjunction with indirect indexed addressing. A minicomputer such as the Texas Instruments 980 or the DEC PDP 11 would have fulfilled the software requirements but not the constraints on size and cost, however a microprocessor could adequately satisfy all these requirements. At the start of this project there were several microprocessors to choose from but only one that had indexed indirect addressing and the speed of operation required, this processor was the MOS Technology MCS6500 an enhancement and improvement of the Motorola M6800 microprocessor.

The development system used in this project had to be designed and built at a component level as funds were not available for the purchase of a commercial development system. The development system consists of two parts, the hardware and the software (or monitor system).

The hardware for the development system consists of a control board which was built at a component level and wire wrapped by hand. This board was necessary since at the start of the project a logic analyser was not available. The control board was constructed using transistor transistor logic (TTL) and is used to control the hardware operation of the microcomputer by using the control signals of the processor. The schematic of the control board is seen in figure 2 and figure 3. The control board allows the processor to be sequenced through the various states of a machine instruction. This operation is used to debug the peripheral hardware of the processor as well as the software.

The functions performed by the control board are as follows; single instruction execution, single cycle execution, breakpoint setting according to a 16 bit address with display and/or stopping of the processor upon detection of the address, display of the R/W and SYNC control signals and finally the trapping and display of address and data bus information.

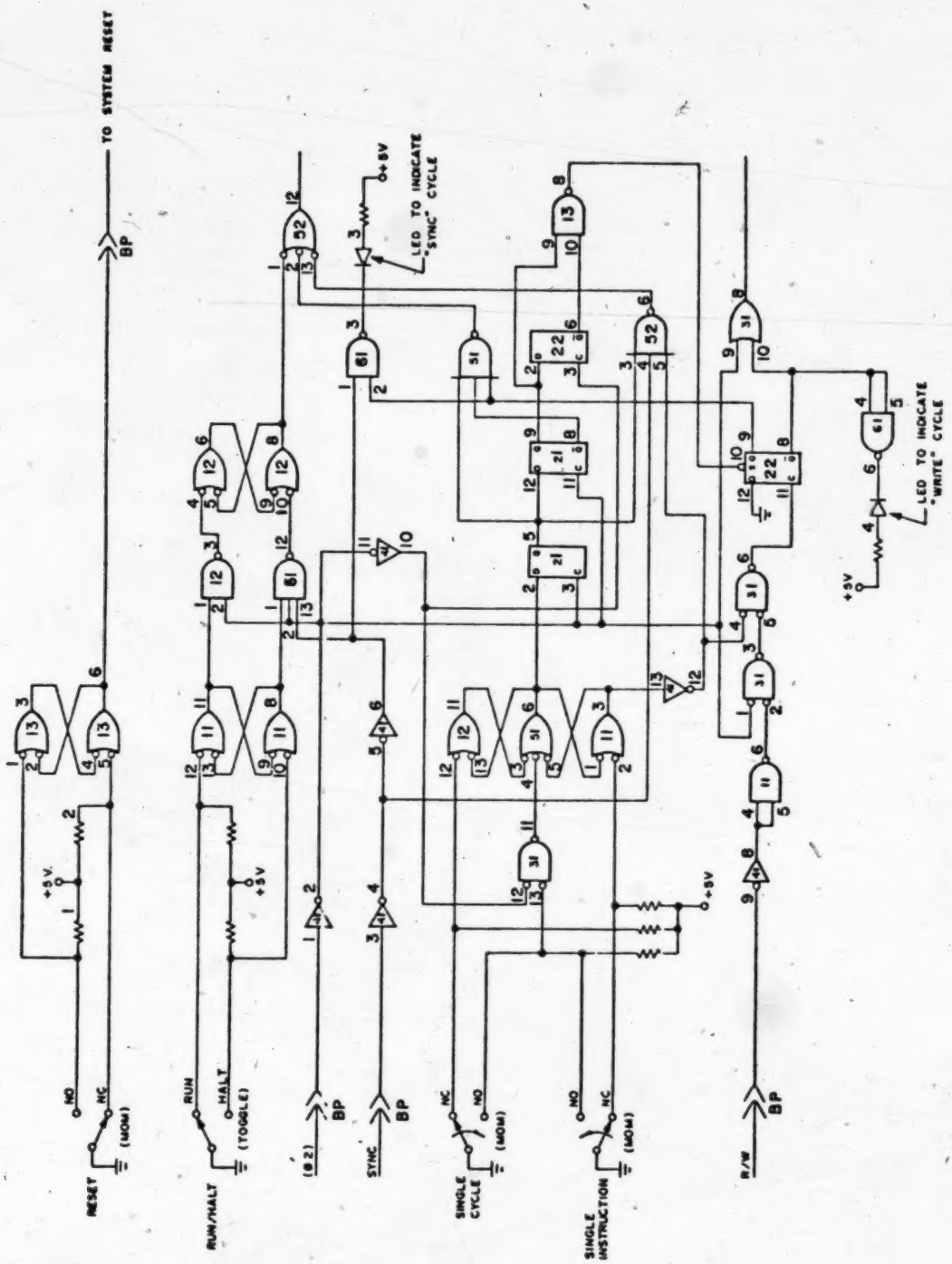

Fig. 2 Control Board - Single Step Logic

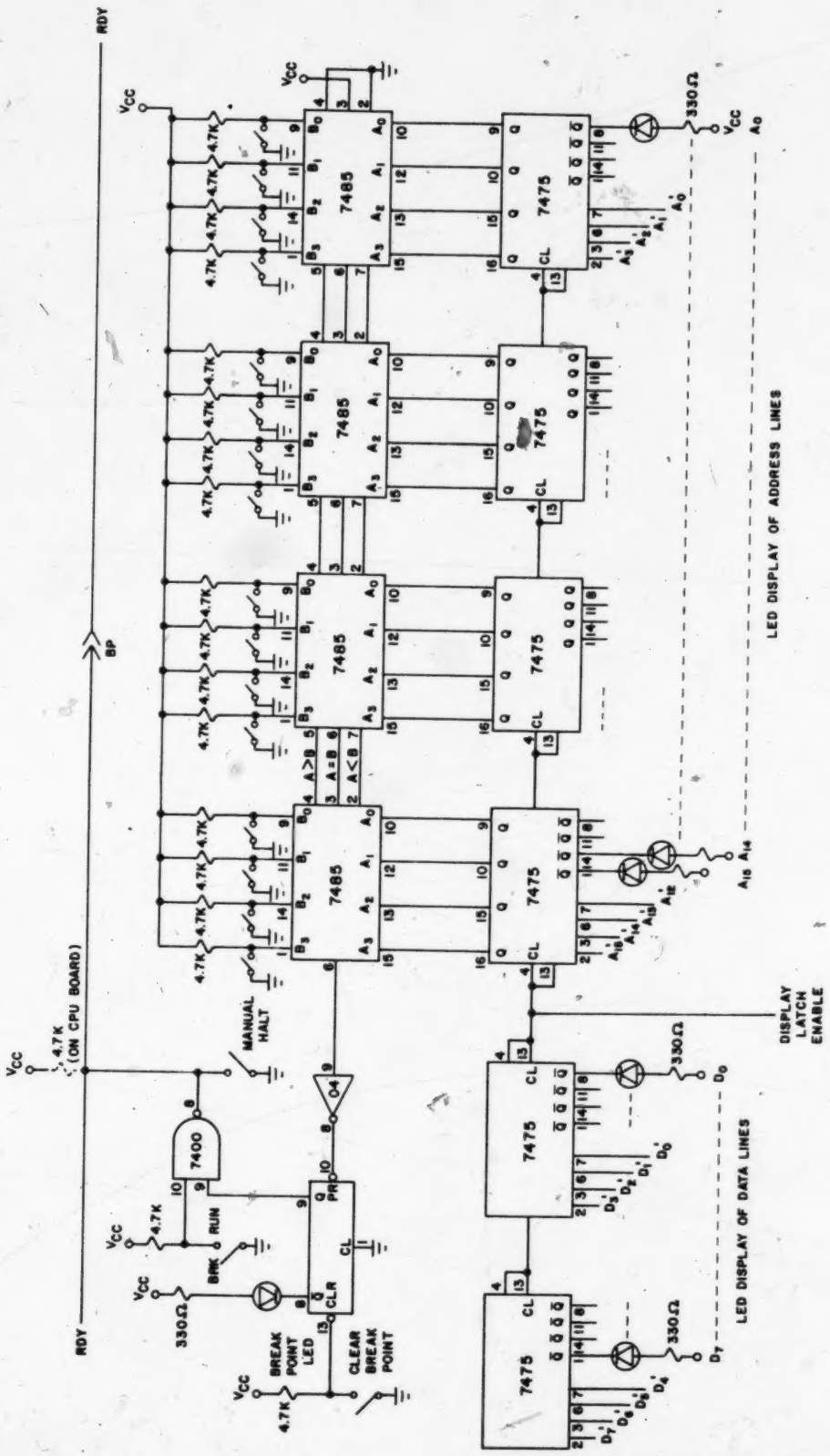

Fig. 3 Control Board - Address Break Point Logic

The development system has a software monitor tb enable the development of 'user' software programs. There are in fact two monitors in the development system; the first monitor was purchased and used to develope the more comprehensive second monitor.

The first monitor program used was in the form of a read only memory (ROM) purchased from MOS Technology. This monitor program is contained on a large scale integrated (LSI) circuit which also contains circuitry for input and output (I/O) as well as an eight bit timer circuit. The device is a MCS6530-004 otherwise known as the terminal interface monitor circuit (TIM). The I/O section of this device can be used to support a video terminal or teletype (TTY).

The initial support for the TIM monitor required only page 0 and 1 of random access memory and a TTY and occupies 1024 bytes (1K=1024 bytes) of memory. This monitor provided only the simplest of software debugging functions such as execute a program, display and change memory, display and change the CPU registers, set software breakpoints (by manually inserting the code for the break instruction) and punch and load paper tape. The programs needed for the Braille algorithms were known beforehand to be complicated and very long and so the TIM microcomputer system was used as a tool to build a more extensive operating and debugging monitor. This subsequent monitor will be referred to as the video monitor (VM) and it uses a cathode ray tube (CRT) as

the human interface.

The VM operating system makes use of a hardware device that interfaces a section of random access memory (RAM) with the CRT screen. This device is on a printed circuit board and was purchased from Polymorphics Inc. The video display unit takes the contents of the video refresh RAM and converts the data to a video data output compatable with a raster scan video monitor. The system assumes that data in the refresh RAM is encoded in ASCII (American Scientific Code of InformationInterchange). The video board is used for two purposes; first it provides a visual display of the text to be transcribed as it is entered by the typist, and secondly it provides a display of various microprocessor operations and functions. The VM is of main interest at this point and will be discussed here. The various functions performed by the text edtor in the transcription mode will be explained later on.

The two separate functions provided by the VM monitor are software controlled. The VM monitor is used to edit and manipulate information in the microcomputer and also to control execution of the 'user program'. The first of these functions is the visual display and/or entry of data into the microcomputer system as a whole; including RAM, I/O, ROM and CPU internal registers. The following visual display and programing aids are done entirely by software and therefore the information appears to the operator instantaneously on the monitor T.V.. The reason for these development functions

are obvious when compared to the limited capability of the TIM and the slow ten character per second (CPS) rate of a standard TTY.

1. The contents of any portion of memory may be displayed and/or altered. The display format is such that one page of information is shown at a time in an 8 byte by 8 line format with the address's vertically shown for each line of data. A cursor is provided so that data can be altered. Full control of the cursor is supported, such as linefeed forward and reverse, forward and reverse spacing and automatic wrap around to the next line and/or page if the current line or page is exceeded.
2. The page preceeding or following the currently displayed page of memory may be selected by a single keyboard command and viewed instantaneously.
3. All the CPU registers may be displayed and altered if desired using the cursor.
4. The contents of memory may be dumped to a cassette tape and read back into memory from the cassette system. The cassette operates at 1200 baud. The main use of this feature is to save programs as they are being developed and to also reload them into RAM for debugging.
5. Block moves of data from one section of memory to another are possible. This feature is primarily used to copy programs contained in ROM to RAM for further debugging or modification.
6. The 'user program' can be executed by setting the

desired entry values into the various CPU registers then typing a single keyboard command.

7. The difference between two 16 bit hexidecimal numbers may be calculated and the result displayed.

This feature is used for the calculation of relative branchs within a program.

The next feature of interest is the control of a 'user program' during execution. It is possible to debug a program using dynamic or static break points. Break points are used in debugging both hardware and software. Encountering a break point in software causes the contents of the CPU registers to be saved and displayed on the CRT. The operation of the program is not altered by the break point, however the break point allows the temperary termination of the 'user program' at any point that is required by the programmer. Break points are set in a program by specifying the break point by a number from zero to fifteen in hexidecimal, and a 16 bit address. Break points may be either static or dynamic and set or deleted individually by number.

Static break points cause termination in execution of the 'user program' and a display of the CPU registers, at which time all the VM functions become available to the programmer.

Dynamic break points allow the execution of the program to be observed at the specified break point locations as the program is running. The program is not stopped when a break

point is encountered, but the CPU registers are displayed for the instruction executed just prior to that break point. The time elapsed between displaying of the current and next breakpoint may be set by the programmer, and so a variable rate of program execution may be observed. The 'user program' while running in this mode may be stopped at anytime by a single entry of any key from the keyboard and may be stopped on any of the break points. Dynamic break points may be changed to static break points by specifying a run rate of zero.

The 'user program' may be sequenced through each instruction of the program one instruction at a time by use of the trace command. This command executes one instruction and then displays the CPU registers and then allows the programmer access to the VM monitor.

II. System Operation

The functional operation of the transcriber is best understood by referring to figure 4. The transcription system consists of four major parts, the video monitor, the Braille printer, the keyboard and the computer system which controls these devices. It can be seen that the transcriber appears to the typist as a computer terminal with a special output device which is the Braille printer.

Several editing features have been implemented to aid in the correction of errors and modification of text. These features are described under Editing Commands.

Special skills are not required to operate the transcriber. The English text is entered into the machine using the keyboard in the same manner that a typist would use a standard typewriter. The text is displayed on the video monitor under a blinking cursor as it is being entered. The cursor is controlled by the computer and is automatically adjusted to the start of the next of the line after the current line is full. The typist does not control the right hand margin of text by using the carriage return as with a standard typewriter; the text is entered in a continous string unless a special format is required. Braille and English text do not generally have the same line length due to contraction, therefore no purpose is achieved by allowing carriage control by the operator. The English text input is never hyphenated for the same reason. There is however one operating feature that is optional; after the

text is typed into the transcriber, the typist may visually search through the text and insert a special non printing character into compound words that are found. This procedure will reduce the number of words incorrectly transcribed by the system. It is however an option and is not critical in the operation of the transcriber.

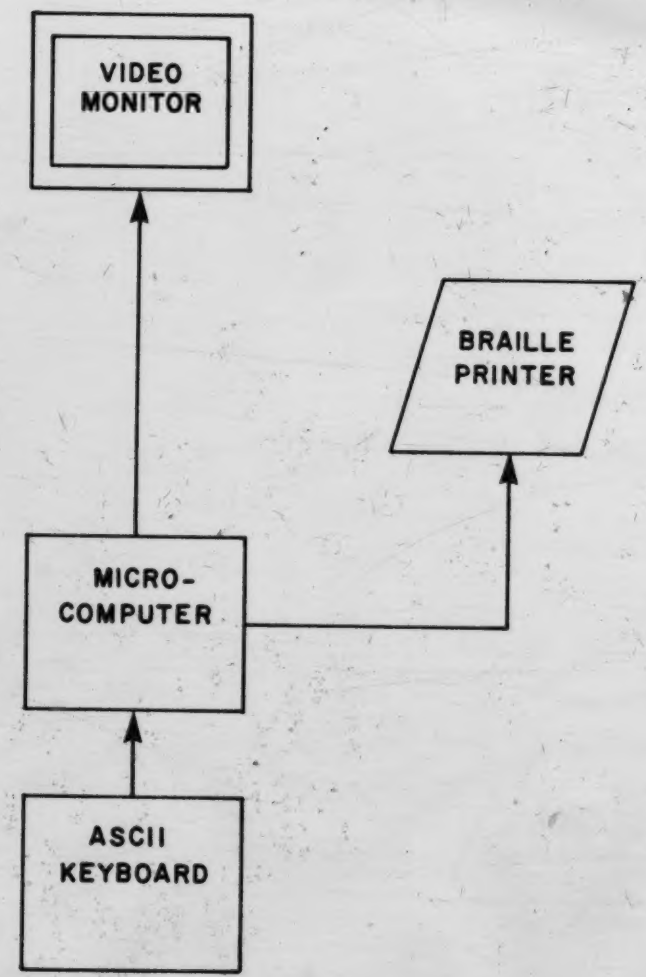

Fig. 4 System Configuration

Transcription Control Command

The following command controls the operation of the computer system and will cause it to transcribe the English text and print the Braille material.

This command is to be preceded by the ESC key:

1. T- Typing this command will cause the computer to remove the current page of text input from the display and transcribe the text at a grade 2 level. As soon as transcription is finished the Braille is output to the printer and the display is cleared. When the blinking cursor appears upon the display the typist may continue entering text; even under worst case conditions this operation never takes longer than a few seconds.

III. System Configuration

The system as shown in figure 5 consists of two parts: hardware, and software. Each part will be described separately.

A. System Hardware

The transcriber hardware is divided into two parts: electronic, and mechanical. The electronics consists of the computer system, the keyboard and the display. The Braille printer is the only mechanical device in this system. A block diagram is shown in figure 5.

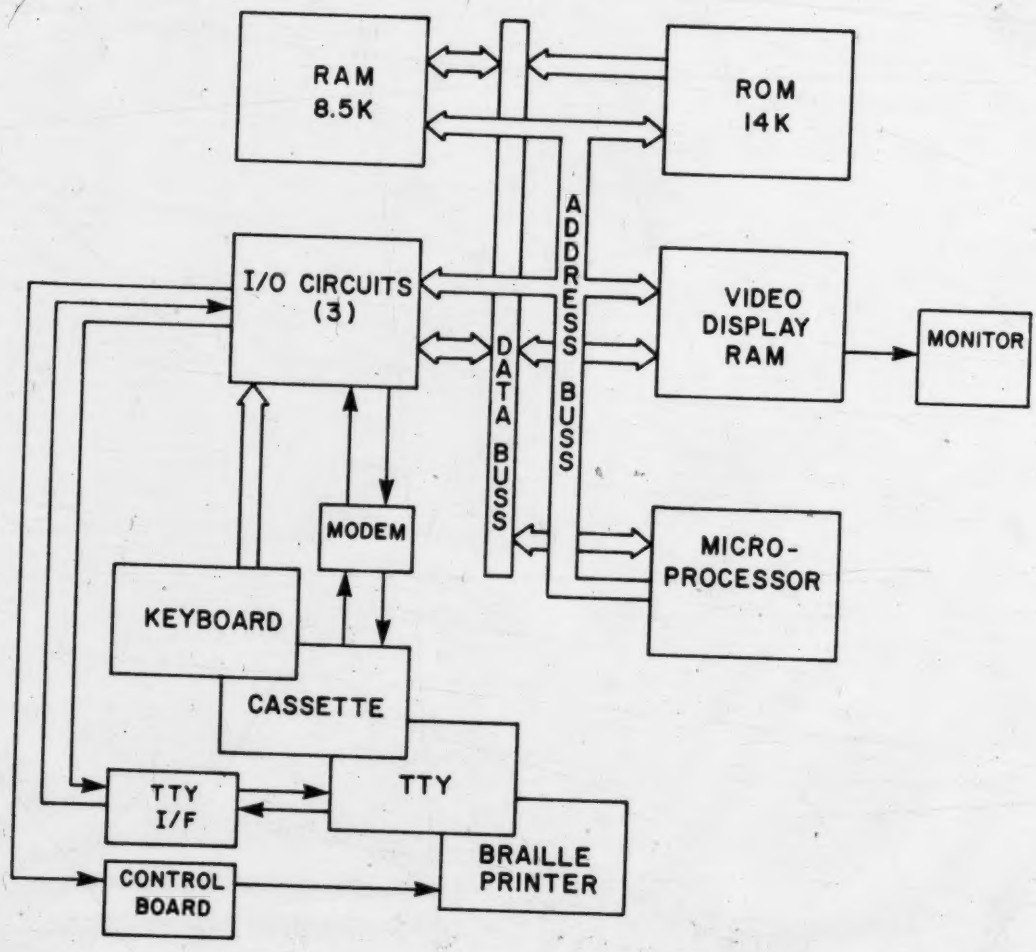

Fig. 5 System Block Diagram

Electronics

The electronics of the Braille transcriber is divided into five units, each of which is assembled on a separate board. These five boards are:

1. CPU and I/O Board
2. ROM Board
3. RAM Board
4. Video Board
5. Solenoid and Motor Driver Board

CPU and I/O Board

The heart of the transcription system is the microprocessor and the support circuits that supply the I/O control. An 8 bit microprocessor was choosen in preference to the 16 bit processors available at the time. The only commercially available 16 bit microcomputer at the start of this project was from National Semiconductor Corporation (NSC) and was termed the PACE 16 (2). The PACE microprocessor has an excellent instruction set but is implemented in PMOS technology which restricts the instruction execution time to between 8 and 13 microseconds. The PACE system requires two power supplies and a large number of support circuits to implement a basic microcomputer. It was therefore decided that the PACE would not be the optimum choice.

There were four main choices in the 8 bit microprocessor area:

1. M6800 by Motorola
2. MCS6502 by MOS Technology
3. 2650 by Signetics
4. 8080 by Intel.

The M6800 (3)is a memory orientated microprocessor. The instruction set of this processor makes heavy use of absolute indexed addressing, with a special mode that uses page 0 as a shortened and compressed absolute addressing form. The primary disadvantage of the M6800 is the primitive indexed addressing form used and the lack of more than one index register. The indexed addressing form used is not true indexed addressing; the instruction can not address the entire memory space with just the code following the opcode. The index register must be used to hold the base address. The main disadvantage of this form of addressing is that, for every indexed reference more than one page away from the last indexed reference, a different pointer must be placed into the index register. Whenever concurrent pointers are needed with indexing a new value must be loaded into the index register. The M6800 is not particularly suited for multidimensional table searchs for this reason. The main advantages of the M6800 is the simplicity of interface, single power supply and an excellent external stack. These advantages however can be found in another processor and therfore the M6800 was rejected as a choice.

The 2650 processor by Signetics (4) has an advanced instruction set for a microprocessor. The addressing modes are excellent and include indirect indexed with auto increment addressing. There are a number of index registers available and register to register instructions may be used with all of the registers. The 2650 also has two distinct banks of registers selectable by a bit in the status register; this feature is used to facilitate interrupt processing. The primary disadvantage of the 2650 is the on board stack. This stack is only 8 levels deep, and severly restricts the nesting of interrupts and subroutines. It is possible to implement an external stack using one of the index registers and some software however, this feature adds a considerable burden on the processing speed of the computer for deeply nested subroutines and interrupts. The 2650 was rejected as a choice primarily on this basis.

The next processor considered was the Intel 8080 (5). This processor is a register oriented machine in that the primary addressing mode for data storage and retrieval is through a pair of 8 bit registers. The register pair is specified in the instruction. This processor does not have true indexed addressing. The register pairs may be incremented and addressing done indirectly through them, but an absolute address may not be added to the memory reference. The 8080 does have an

excellent external stack. The microcomputer as a whole is however, more expensive and complicated to implement because addition circuits are required to demultiplex the data bus and derive the system clock. The 8080 also requires 3 power supplies. The 8080 was rejected on the basis of inadequate software structure and the added complexity of the extra support circuits needed to assemble a basic microcomputer.

The microprocessor chosen for this project was the MCS6502 by MOS TECHNOLOGY (6). This processor is an enhancement of the M6800 by Motorola. The MCS6502 is an 8 bit microprocessor capable of addressing an address space of 65k memory locations. There are two versions of the MCS6502 processor available, one has a maximum clock frequency of 1MHZ. and the other a maximum clock frequency of 2 MHZ giving a minimum instruction execution time of 2 and 1 microsecond respectively. The 1MHZ version of the MCS6502 was used for this project primarily because the memory and I/O parts where the easiest to obtain for the 1MHZ version. The MCS6502 was the first processor to make use of an on-board clock and as well the processor operates from a single +5 volt power supply. The family product line of circuits for the M6800 are directly compatable with the MCS6502. The MCS6502 may be stepped through single cycles of an instruction unlike the M6800; this feature was incorporated in the hardware debugging board.

The programing model of the MCS6502 is shown in figure 6. The MCS6502 (7) has two 8 bit index registers X and Y which may be used with the indirect addressing capability of the MCS6502. These registers offer true post and pre-indexed indirect addressing in page 0. Page 0 is reserved for a short compressed form of absolute addressing and also for indirect addressing. This indirect addressing allows the use of up to 128 (decimal) pointers and coupled with its indexing capability makes this machine ideal for multidimensional table searchs. Although the processor only has 8 bit index registers this does not represent a problem, since the higher order byte of the indirect pointer may be adjusted by one each time the index register overflows. The processor also has an external stack in page 1. The stack is fixed at page 1, but again this is not a problem since it is possible to nest subroutines and interrupts up to about 80 (decimal) deep. The stack is not used to store large strings of data. Data storage is done indirectly through page 0 pointers. The processor has only one accumulator; this is usually sufficient since most of the operations in the Braille programs require a variety of data movment and compare instructions and relatively few calculations.

The MCS6502 microprocessor has 3 levels of vectored interrupt. They are RESET, NMI(non-maskable interrupt), and IRQ(interrupt request). In addition to these levels

a software interrupt can be implemented as a fourth level of interrupt. The software interrupt or break (BRK) instruction shares the same hardware vector as the IRQ vector; a bit is set in the status register to signify the BRK interrupt.

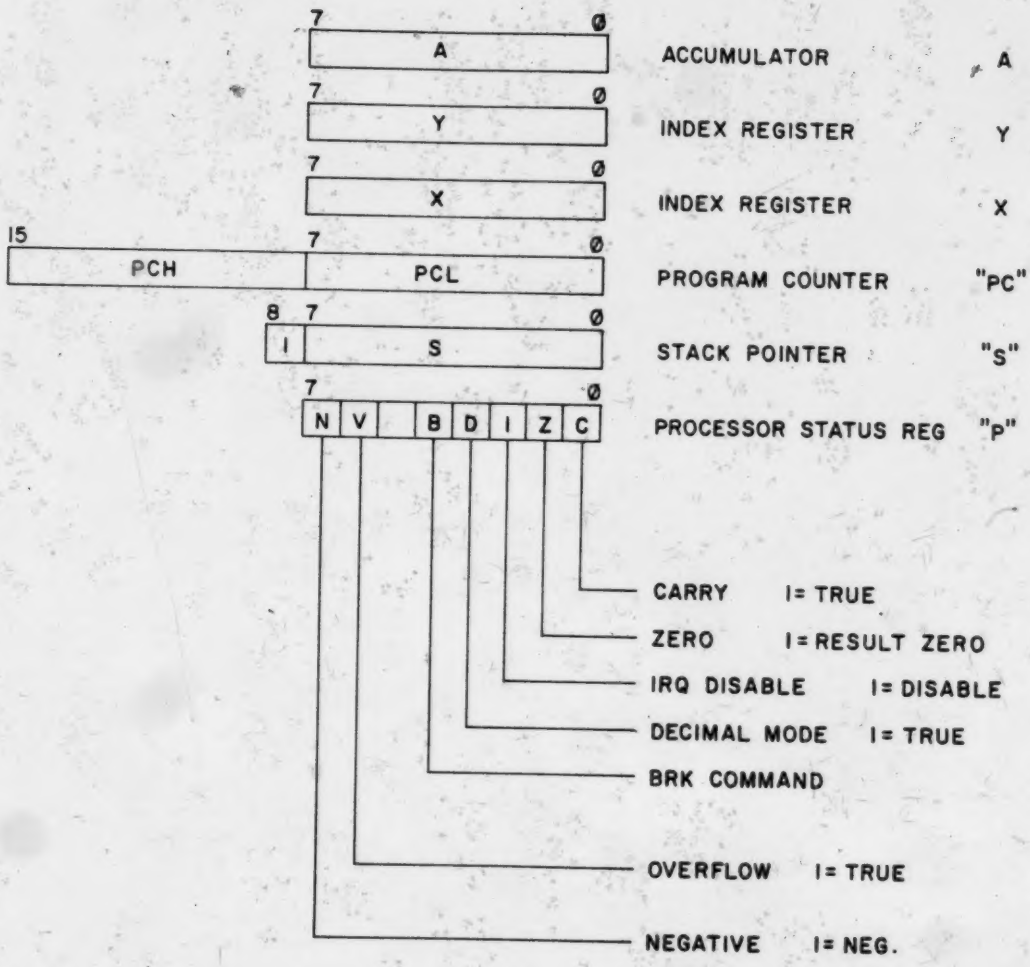

Fig. 6 Programing Model

The CPU board was built at a component level using wire wrap sockets and standard vector board. A schematic of the CPU board is shown in figures 7 and 8. The Braille transcriber section of the CPU board comprises only part of the CPU board. Before the assembly and debugging of the actual Braille transcriber programs could begin, a working microcomputer complete with its own monitor system had to be constructed. The first monitor system used a manufacturer supplied ROM to operate a TTY. This monitor system may still be accessed although it is no longer used. This 'Terminal Interface Monitor' was then used to construct a more sophisticated monitor system. This new monitor allows the microproccessor to be used with a video display. This system monitor was also constructed to control break point operation and visual execution of the programs under development directly from a keyboard. The programs for the transcriber were developed using this monitor system.

The CPU and I/O board consists mainly of ROM, RAM,I/O and support circuits. The transcription programs occupy approximately 11K locations. The video monitor program is 2K locations long. Page one and page two are unique to the software and hardware architecture of the MCS6502 and have been implemented in RAM directly on the CPU board. This feature allows the monitors to be used without the extra memory boards in the system. The I/O

ports used for the Braille transcriber section is the newer version of the M6820 which is the MCS6522 and is hereafter referred to as the VIA (verstile interface adapter). However, one M6820 (peripheral interface adapter) is used, to connect the ASCII keyboard to the system for use by the video monitor. The video monitor still uses the PIA for this purpose however the transcription system uses the same ASCII keyboard connected to a VIA.

The support circuits used consist primarily of decoding logic, address and data buffers, and TTL circuits for combinational logic.

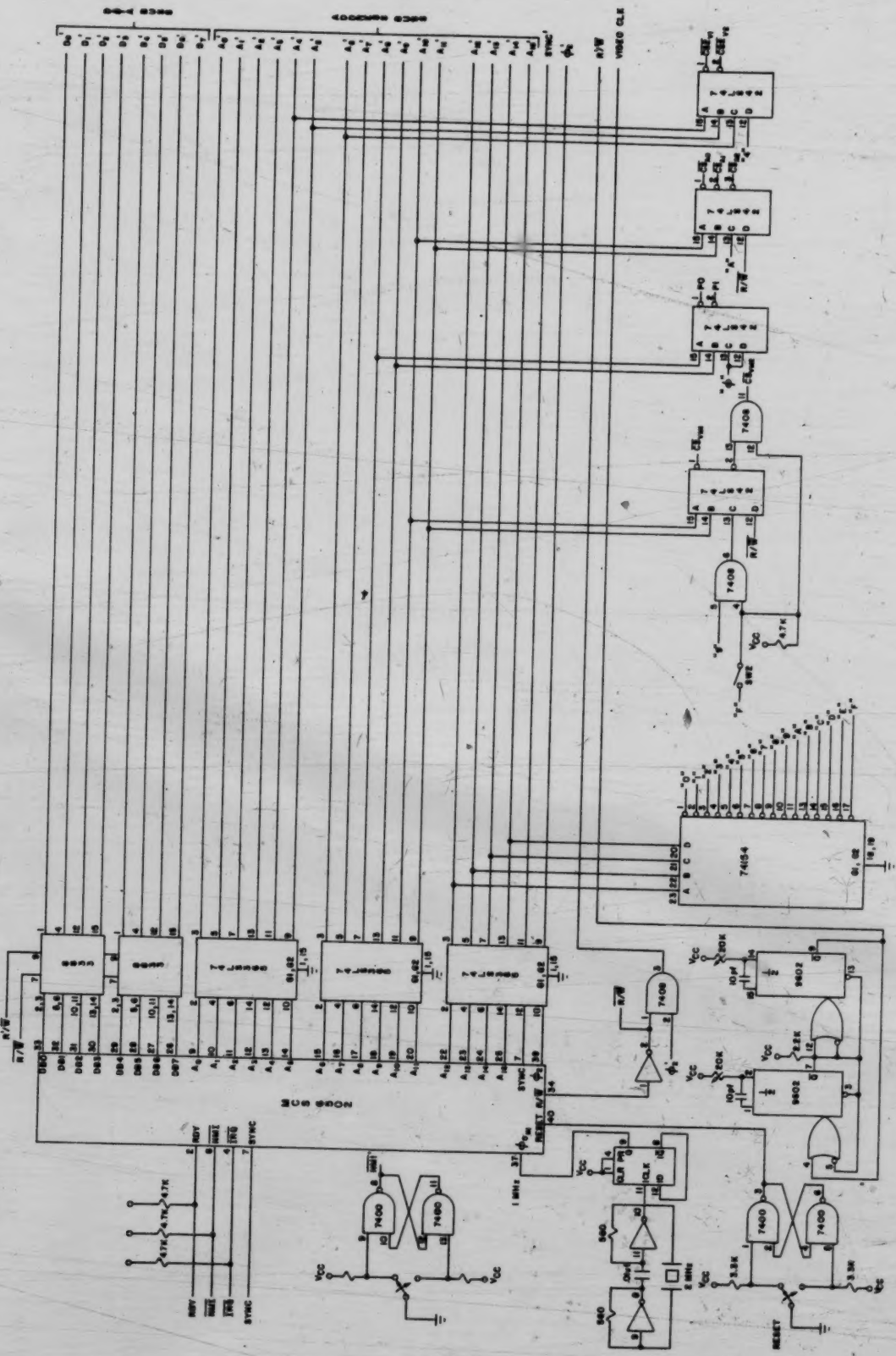

Fig. 7 CPU Board --Processor and Address Decoding

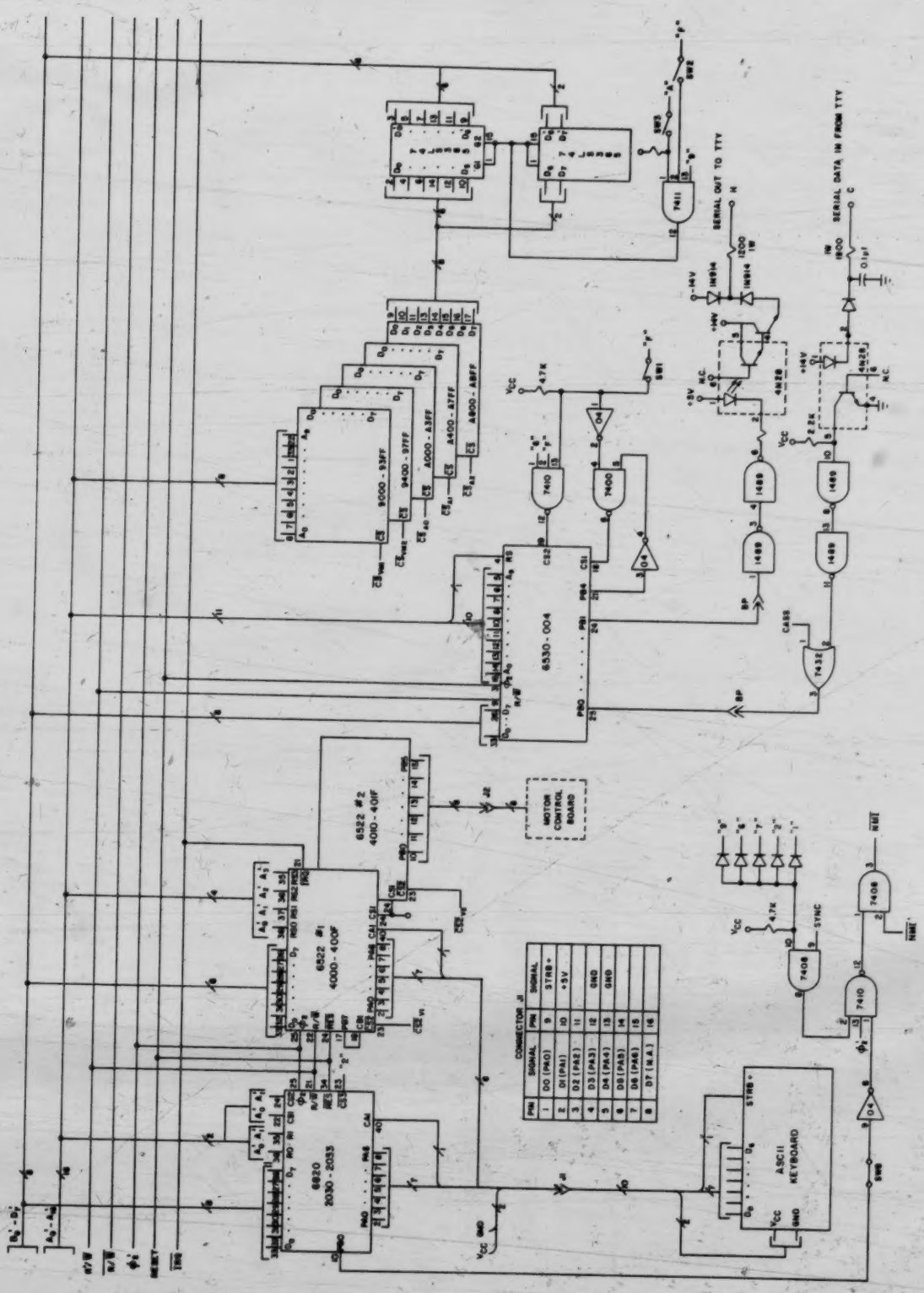

Fig. 8 CPU Board -I/O and ROM

ROM and I/O Board

The ROM and Cassette board was built using wirewrap terminals and vector board, as was the CPU board, the schematic of the ROM and Cassette board is shown in figure 9. The ROM portion of the board contains 11K words of memory, assembled from 1kx8 bit memories. Since this board is read only, data will travel in one direction only, that is from the memory to the CPU. These lines are buffered, as are the address lines.

The address space that is occupied by the ROM is fixed to adresses A000-CFFF and this address space is decoded into unique 1k slots of memory. This ROM may be deselected from the address space in 4k blocks. The two RAM boards each contain 4k of memory, and may be positioned anywhere within the 65k address space. The RAM boards may occupy the address space of the ROM if the ROM has been deselected. This feature was used in the development of software programs before committing them to ROM. The RAM boards also have the capability of memory write protection. When the memory protect switch is set the data in RAM can not be altered. This feature allows the RAM to appear as ROM to the software.

Programing the ROM's has been done on a microprocessor development system. The development system that was used was the American Microsystems Incorporated (AMI) development system for the M6800 microprocessor. The AMI development system has the

capability of taking the contents of development system memory and programing this information into ROM. The AMI development system supports an editor/assembler for the M6800 but not for the MCS6502, therefore the code for the Braille programs were assembled by hand and the machine code entered into the AMI RAM. A special program was then run to program the EPROM's.

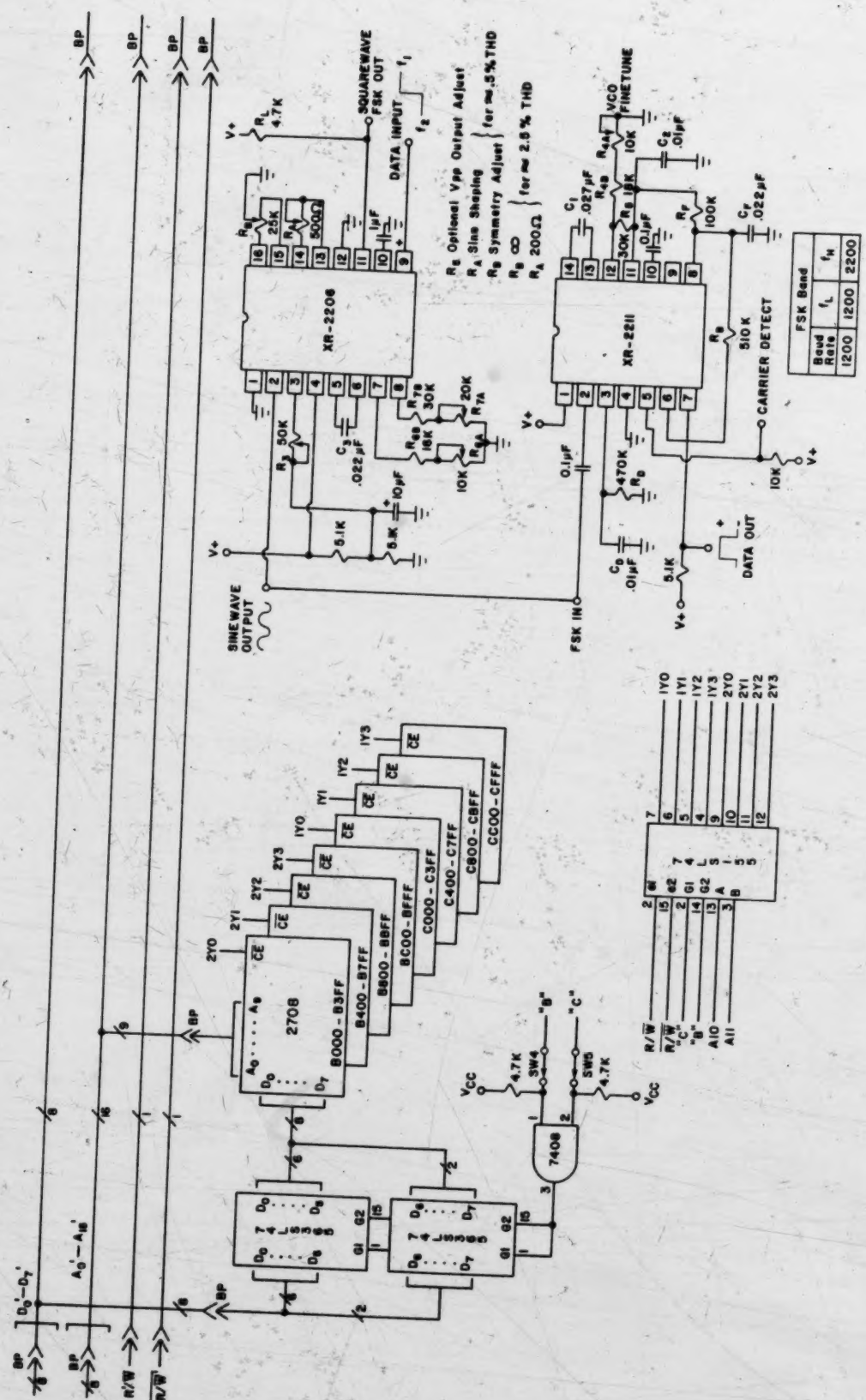

Fig. 9 RCM and Cassette Board

RAM Board

The random access memory is used by the Braille transcription programs to maintain a copy of the text information as it is being transcribed and to hold the information to be printed. Programing variables are also retained in RAM for ease of modification. Although the actual Braille transcription programs reqire only 2.5k of RAM about 8.5k was imlemented, most of which has been necessary for the developement portion of the microprocessor system.

Two identical printed circuit boards were purchased and used to provide the system RAM. Each of the boards is 4k words in length and each have been modified to work with the MCS6502 CPU. The schematics of the RAM boards may be seen in figures 10, 11, and 12 (8). Originally the memory boards were designed to work with a 8080 CPU using a split data bus and 2102 memories which have separate input and output data lines. The memory systems were modified to provide bi-directional data flow on a single data bus; appropriate timing signals were created from the MCS6502 signals to interface to the 8080 timing signals required by the memory board. This arrangement provides 8k words of RAM with an additional 0.5K of RAM on pages one and zero located on the CPU board. Buffering is provided for the data and address bus on the RAM boards.

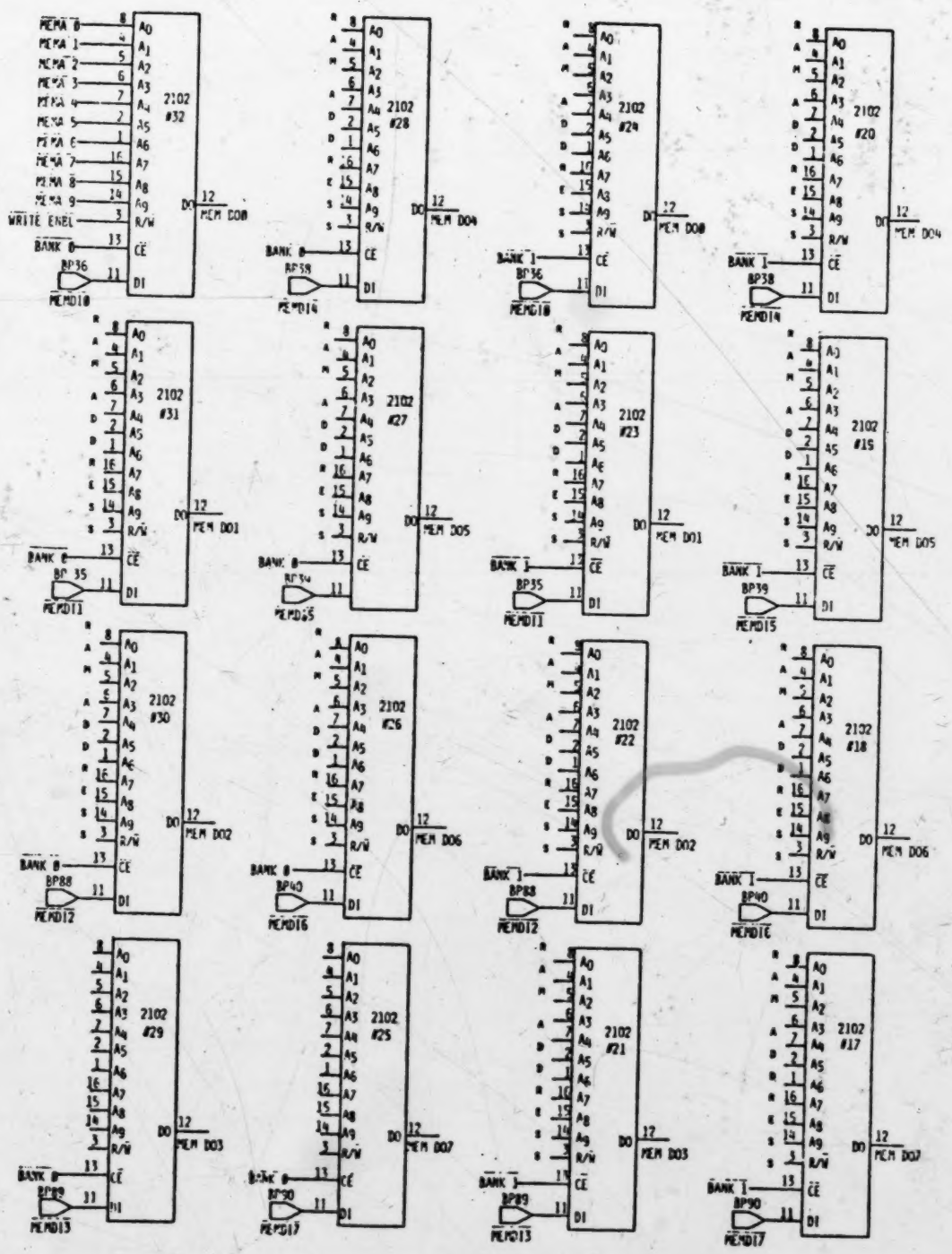

Fig. 10 RAM Board -Bank 0 and 1

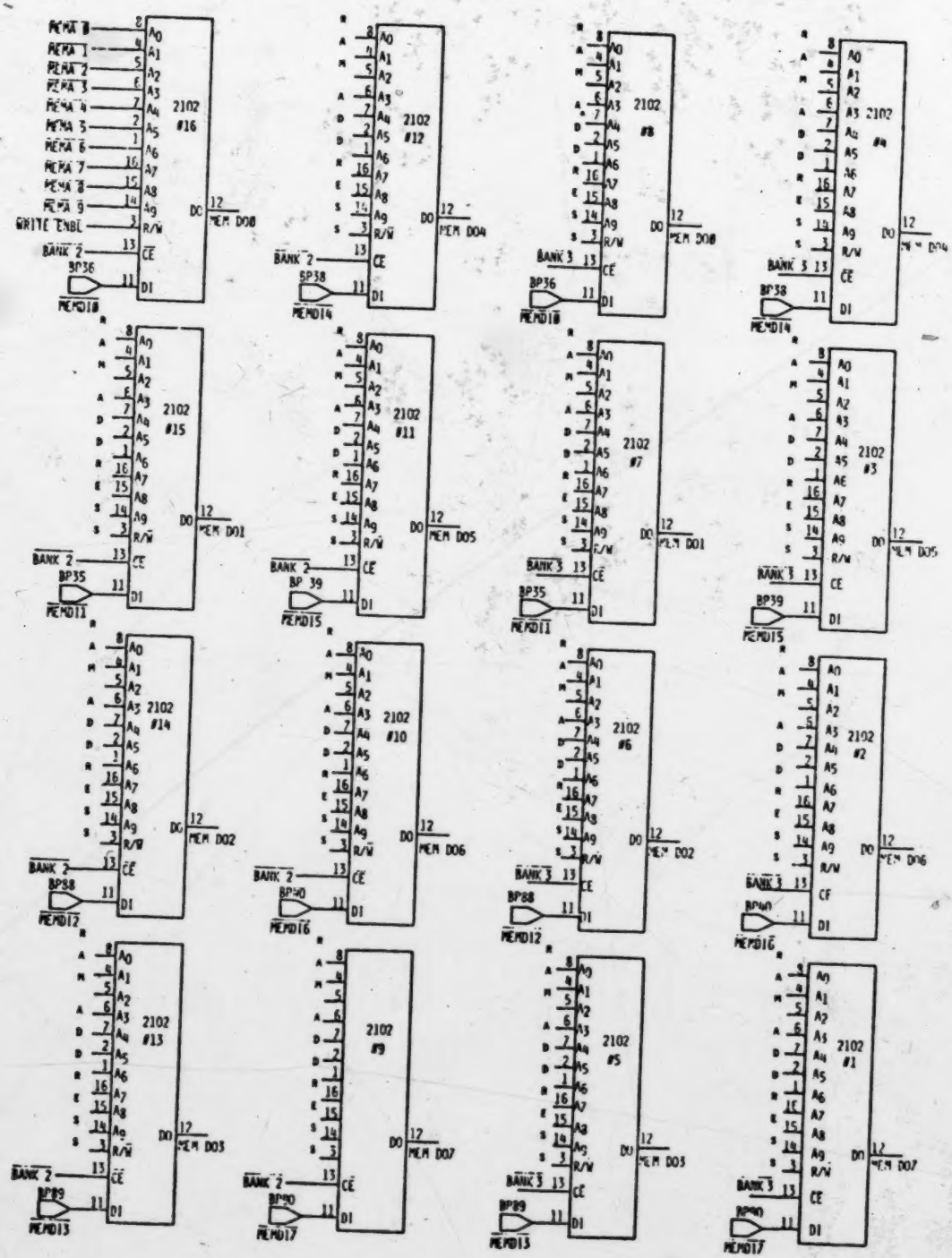

Fig. 11 RAM Board -Bank 2 and 3

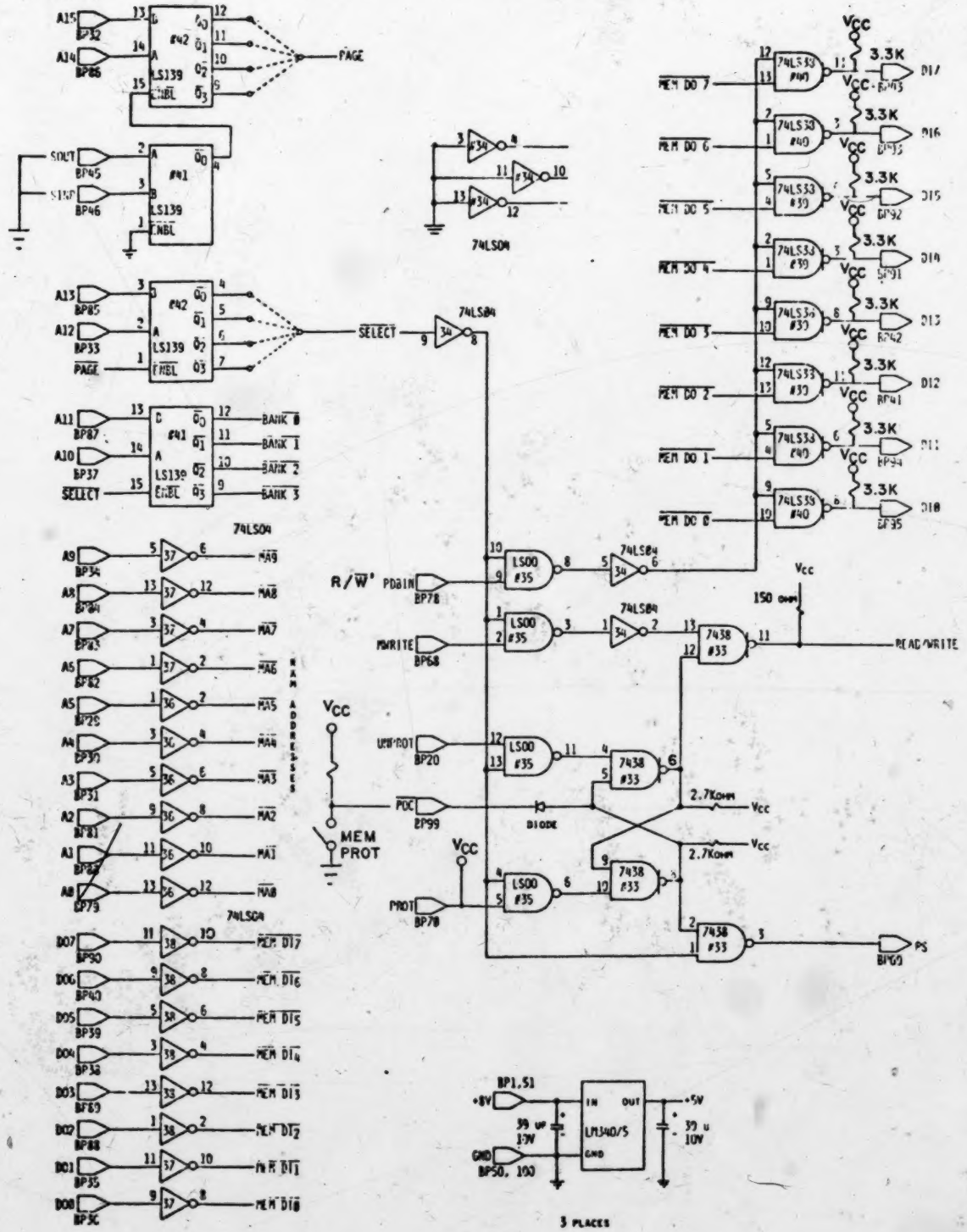

Fig. 12 RAM Board -Address Decoding and Drivers

Video Board

The video board is used to display the text as it is entered by the typist. The board appears to the CPU as a standard block of decoded memory space, with the usual timing signals. To display a character it is stored at a location in the memory system which corresponds to its location on the screen. There are 1k locations of 8 bits in the video board. To display the standard English alphabet and number system the ASCII representation of each symbol is stored at the required location.

The schematic of the video board is seen in figure 13 (9). The video board operation is similiar to most modern computer terminals; information placed in RAM is in ASCII code, the code is accessed as data output from RAM and is supplied as an address to a character generator. The row outputs of the character generator are strobed into a shift register. The shift register is clocked to provide a serial data stream that coincides with the sweep rate of the CRT used. This information is combined with counting circuitry outputs to provide a composite video output, which drives the CRT. The address information to the RAM, previously mentioned, is multplexed between the external CPU address bus and the internal refresh counter circuitry. The CPU is given priority when timing conflicts.

The board is fully buffered for both the data and

address buses. However, as with the RAM boards, the original design of the video board was for a 8080 CPU with a split data input and output bus. The video board has been modified to drive a bi-directional single data bus. Again the timing signals from the MCS6502 have been modified to this board to provide timing signals which are compatabile with 8080 timing signals.

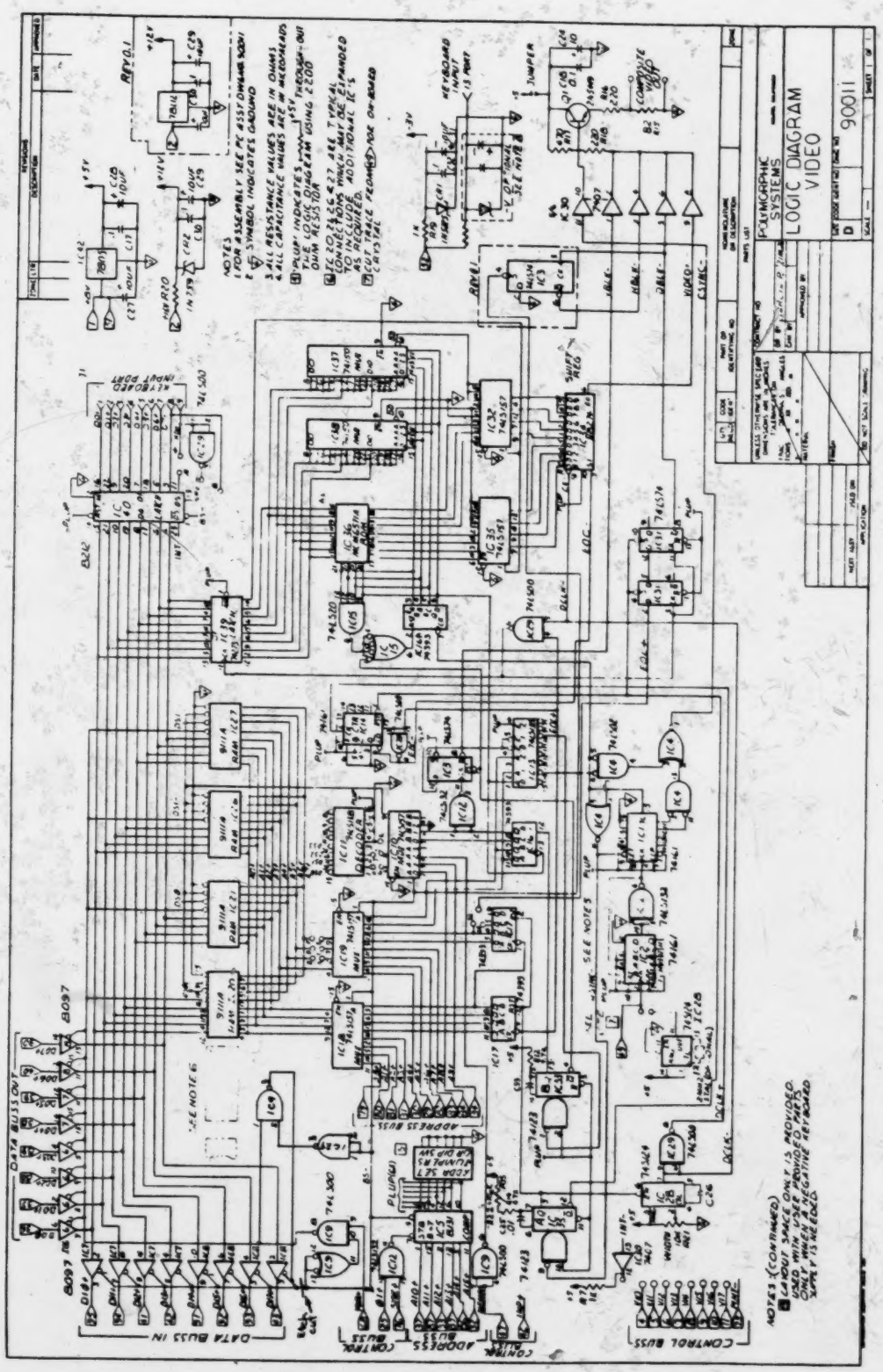

Fig. 13 Video Board -With Modifications

Solenoid and Motor Driver Board

This board is used to control the two stepper motors and the three solenoids used in the printer, the schematics may be seen in figure 14. The stepper motors used, provide an angle of rotation of 15 degrees for each step. The motors are D.C. motors and are bi-directional. A linear integrated circuit made for controlling these particular stepper motors, controls the correct switching sequence and switchs up to 700 milliamperes of current per winding. There are four windings per motor and the direction of rotation is determined by the sequence in which the windings are energized. By applying a single strobe to the trigger input of this circuit, the correct switching sequence is initiated. Direction of rotation is determined by a level setting at one of the inputs to this circuit. All of these signals are controlled by the VIA under program control; operation of these signals will be explained under the software section. Since there are two stepper motors two such circuits have been provided.

Three solenoids are used to emboss the paper and are of the push type. Each solenoid draws about 0.75 amperes. A Darlington transistor configuration is used to drive each solenoid which is, in turn, driven by the VIA output.

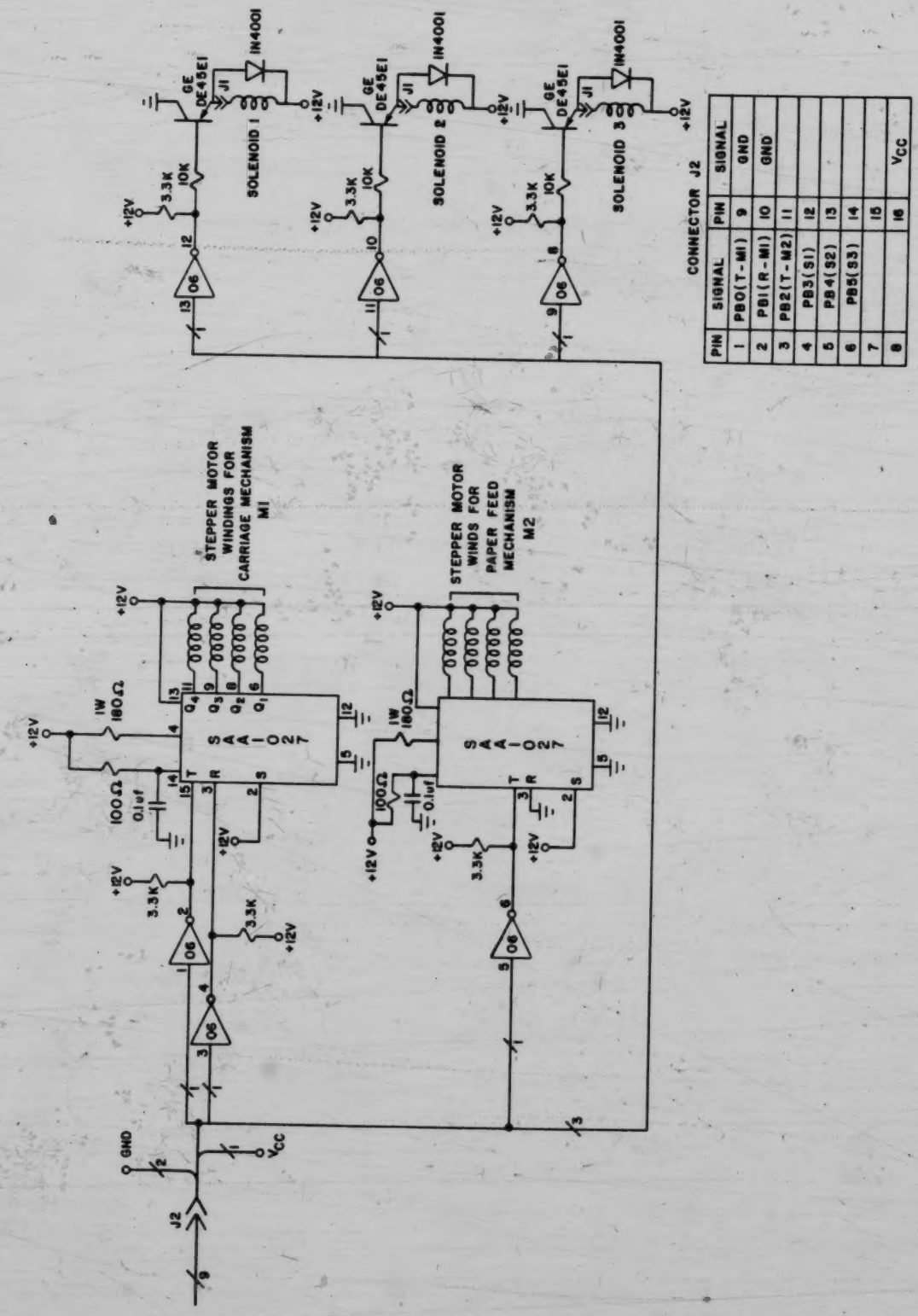

Fig. 14 Solenoid and Motor Driver Board

Mechanical

The construction of the braille printer is shown in figure 15. It can be seen that the Braille printer consists of four major sections: the chassis, carriage mechanism, roller assembly and embossing mechanism.

The chassis has been made out of sheet aluminium with aluminium spacers for the frame work. The carriage mechanism is driven by a belt and sprocket on a pair of guides. The paper roller is rubber, mounted with bearings and is used to linefeed the paper. The paper guide is held with spring tension against the paper roller and when the roller is rotated the paper is fed through the mechanism due to friction. Both the carriage mechanism and the roller are driven by stepper motors. The solenoid block is mounted to the carriage mechanism and contains the three solenoids used for emobossing.

The electronics used to drive the printer has already been described and the software programs needed to produce the Braille output will be discussed later in the software section.

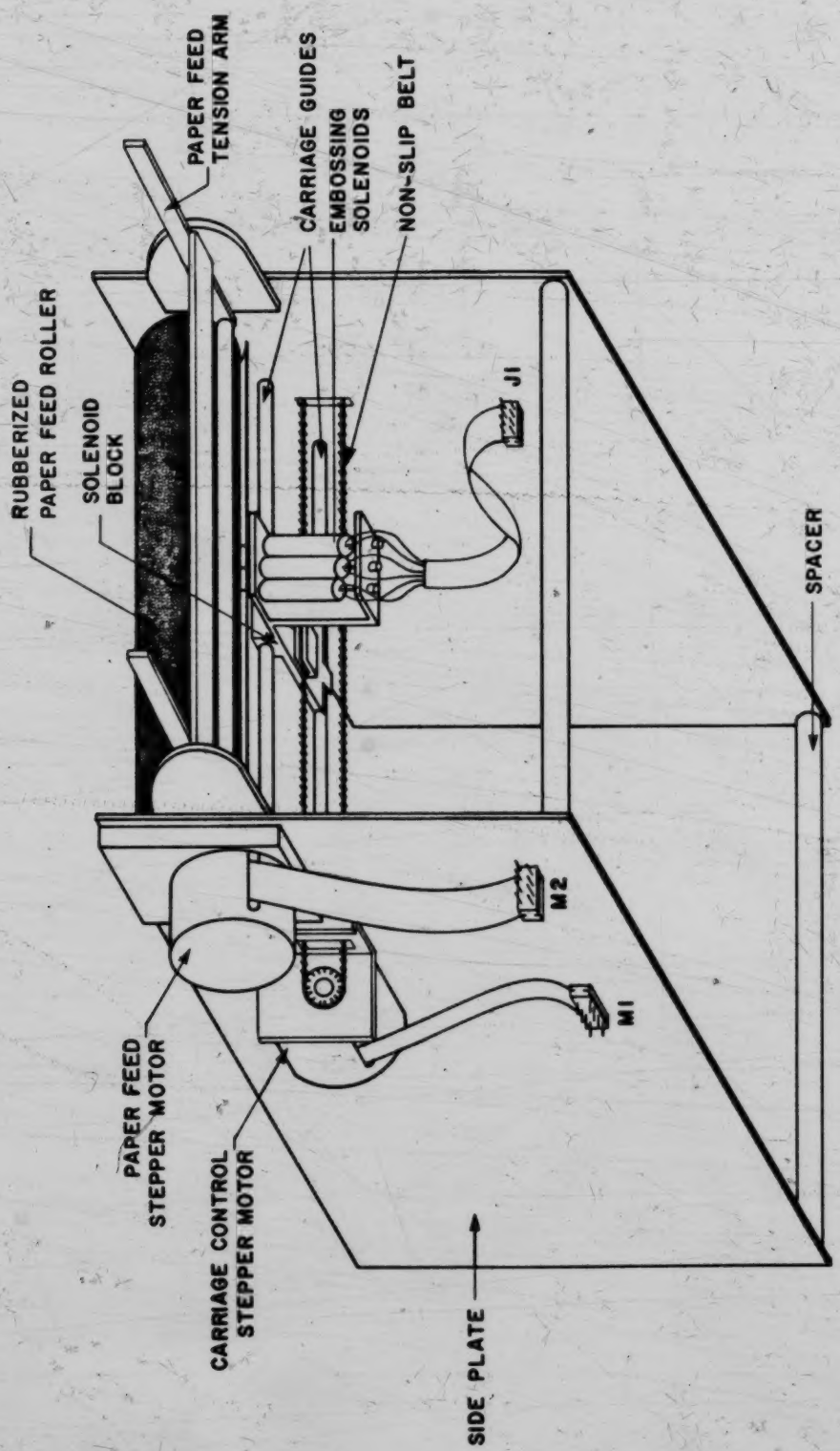

Fig. 15 Braille Printer

B. Software Structure

The main reason the MCS6502 microprocessor was chosen for this project is its programing architecture. Figure 6 shows the programing model of the CPU, it can be seen that the machine has two 8 bit index registers X and Y, a single 8 bit accumulator A, an 8 bit stack pointer S, a 16 bit program counter PC, and an 8 bit status register P. The MCS6502 has 13 different addressing modes that can be used to manipulate data. The following is a list of the addressing modes:

1. accumulator- is a one byte instruction operating on the accumulator.
2. immediate- the operand is contained in the second byte of the instruction.
3. zero page- the second byte of the instruction is an effective address in page zero.
4. zero page,X- the effective address in page zero is indexed using the X index register.
5. zero page,y- the effective address in page zero is indexed using the Y index register.
6. absolute- the effective address is contained in the second and third bytes of the instruction.
7. absolute,x- the effective address is formed by adding the X index register to the second and third bytes of the instruction.
8. absolute,y- the effective address is formed by adding the Y index register to the second and third bytes of

the instruction.

9. implied- one of the CPU registers is implied.
10. relative- the effective address is formed by adding the 2's complement of the second byte of the instruction to the present value of the program counter.
11. (indirect,X)- the second byte of the instruction is added to the X index register discarding the carry and the result points to a location in page zero. The contents of this location is used as the effective address for this instruction format. This format is referred to as post-indexed indirect addressing.
12. (indirect),Y- the second byte of the instruction points to a location in page zero, the contents of this memory location are added to the Y index register to form the effective address. This format is refered to as pre-indexed indirect addressing.
13. absolute indirect- the second and third byte of the instruction contains an address. The contents of this location is the effective address.

The indirect indexed addressing capability is necessary for the the search routines and file management routines. Without this addressing capability, tables of index register values would have to be stored in ROM and this would slow execution of the system through put as well as increase the physical size of the program.

Figure 16 shows the memory map for the computer. The software for the transcriber is written in seven separate

software modules. Each of these modules will be discussed individually. The seven modules are:

1. Initialization
2. Interupt Decoding
3. Control
4. Text Editor
5. Support Routines
6. Transcription Routines
7. I/O Routines.

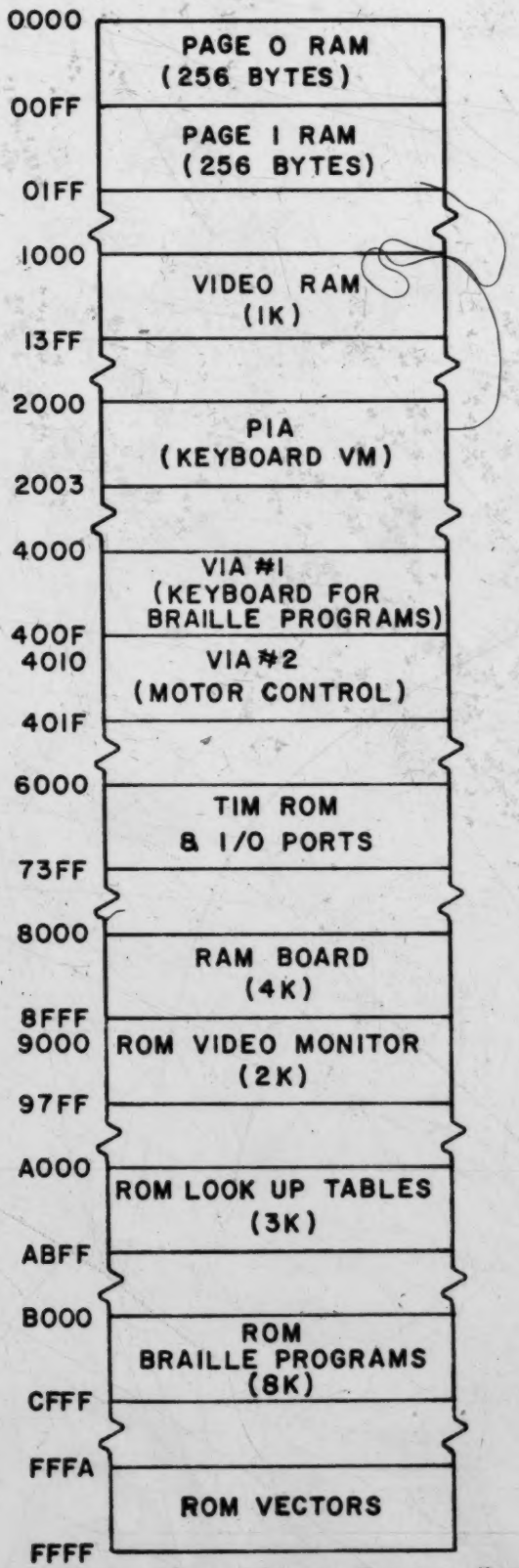

Fig. 16 Memory Map

Initialization

Hardware and software interact to prepare the computer system for operation. All of the following hardware actions occur before software execution begins. At reset the CPU fetchs the reset vector and begins program execution at the specified address. Maskable interrupts are automatically inhibited until enabled under program control. Non-maskable interrupts are not disabled since as the name implies there is no mask to disable them. Since this system does not make use of the non-maskable interrupt (NMI) no further action is necessary with respect to the non-maskable interrupt. Reset also causes the initialization of the interface peripherals connected to the reset line. The verstile interface adapters (VIA) are initiallized to prevent interrupts from occuring until the software deliberately allows the required interrupt. The VIA may interrupt because of several conditions: for example, a transition on one of the control lines, the timing out of one of the two onboard counters, or a specified condition of the shift register. The VIA has 16 addressable memory locations within the device. There are 4 registers which control the operating mode of the VIA, these registers are the auxiliary control register, the peripheral control register, the interrupt flag register, and the interrupt enable register. The interrupt enable register allows or disallows interrupts from the ports, timer/counter pair, shift register and control lines depending upon the setting of bits in this register. The interrupt flag

register flags all interrupt conditions by setting bits and will cause the VIA to bring low the interrupt line to the CPU if enabled to do so by the interrupt enable register. The interrupt line will remain low until recognized by the CPU. The peripheral control register sets the operating modes for the A and B ports and their control lines. The auxiliary control register sets the operating modes of the remaining registers.

The CPU begins execution of the software by setting the stack pointer to the top of page one. The stack pointer of the MCS6502 is fixed to page one and may address 256 locations. The next operation to be performed is the software initialization of the VIA ports. The keyboard is connected to the 'A' side of one of the VIA's and the key strobe is connected to the CA1 control line. The peripheral control register is set to detect changes on the positive rising edge of the control signal connected to the key strobe. The auxillary control register is set to allow data support on both A and B ports. The appropriate bit is set in the interrupt enable register to allow an interrupt to occur on the transition of the keystrobe signal. There are several bits in the interrupt flag register each controls a particular hardware interrupt condition. This is all that is needed for the key board initialization.

The blinking cursor is controlled by time-outs of a counter which will cause an interrupt to the CPU. The timer/counters one and two are initialized by setting, in

the auxillary control register, the mode in which they will operate. The timer/counters may operate either as timers or as counters. Timer one is set to operate in the free running mode (the timer is automatically reloaded after a time-out, and a continuous square wave generated), counter two is set to count the square wave outputs from timer one by decrementing the counter two register. When zero has been detected a bit will be set in the interrupt flag register. The interrupt enable flag register is set to allow interrupts on counter two time-outs.

The B side port of a VIA is used to control the stepper motor. Again the remaining timers are similarly set to produce a real time interrupt under control of the software for control of the motor response time. The above completes initialization of the VIA ports.

The only remaining parameters to set are special program constants, and pointers. Pointers are now set to point to the first location of the video display area. The control word for the punctuation routine is reset and finally the hardware counters are initialized to count. Interupts are now enabled and the program waits for an interrupt to occur.

Interupt Decoding

The program used for interrupt service is of the polled interrupt type. All the devices requesting service are slow (in the order of milliseconds) and a polling routine works efficiently when coupled with the excellent interrupt

facilities of the VIA.

All of the peripheral devices are interrupt driven, and always by one of the VIA. In addition to these interupts, real-time interrupts occur for the blinking cursor and the solenoid and stepper motor time-outs.

The hardware devices that require interrupt service are the keyboard, solenoids, and stepper motors. As soon as an interrupt is recognized by the CPU the program vectors to the interrut decoding routine. This program tests the VIA interrupt registers to determine which device caused the interrupt. The particular VIA from which the interrupt occured will have bit 7 set equal to a one. The MCS6502 has a particular instruction, the BIT TEST instruction, which will set flags in the status register to represent the status of bits 6 and 7 at the location tested. This instruction is followed by the appropriate branch and the interrupting VIA is found. The interrupt register of the VIA is polled to find the interrupting device and a jump executed to the relavent routine. After the particular port (or timer) is read, the interrupt is automatically reset. It is possible to not clear the interrupt if so desired however, this approach has not been needed and therefore is not used.

Control

This routine is interrupt driven by the keyboard. As soon as a key is depressed an interrupt is generated and the interrupt decoding routine passes control to the control

routine. The primary function of the control routine is to direct program execution to the required program section.

The keyboard port is read and the ASCII input decoded as to the function to be performed. If the input is the ESC key a wait loop for another input from the keyboard is implemented and the interrupts are left disabled. If the input is a control code a text editor function is executed. The editor functions will be explained under the topic 'Text Editor'. If the input is any symbol other than a control or ESC code it is stored on the screen and the pointers to the cursor adjusted. A return from the interrupt routine is now implemented and the interrupt are enabled again.

Text Editor

The text editor allows for the addition and change of text displayed on the screen. Many of the text editor commands are invoked using a control character, and the remaining commands must be preceeded by the ESC character.

An indirect pointer is maintained in page zero to point to the characters on the video display. Editing functions such as line feeding the cursor are easily implemented by adjusting this video pointer using addition or subtraction of one line of character positions. The same operations are implemented to forward or backspace the cursor, except that one character position is added or subtracted from the pointer. The remaining cursor movements of multiple character positions horizontally and vertically are similarly implemented by appropriately adjusting the video

pointer.

Commands are allowed for the clearing of text from the screen by line, page and to the end of the page. These functions are implemented by calculating the start and end vectors of the area to be cleared and then storing the ASCII code for a space on the screen between the limits of the pointers.

The remaining set of text editing routines have to do with the insertion of and deletion of text from a line. Each function will be considered separately.

The insert command allows a string of characters to be inserted a cursor position. The text following the cusor is moved forward one position for each character inserted. The next open position on the screen is moved one position forward as each character is entered. This operation maintains the proper right to left manner of entery common to English. The software needed to implement the above command requires a block move subroutine and a subroutine to adjust the video pointer by plus one. Start and end vectors are calculated by the block move subroutine and the text following the last inserted character is moved forward one character position. The program will remain in this mode until the 'control O' code is entered by the typist. When the above code is received the keyboard interrupts are enabled and return is made to the main routine. The delete mode is similiar to the insert command except that the cursor is not moved and the text is adjusted from right to

left. The delete function may be used with the repeat key if desired. The text is block moved in reverse from the current entry point to the end of the page.

Transcription Routines

The main body of software in the Braille transcriber is contained in the transcription routines. These routines are structured into five major parts with each of the parts having further sub-division. Each part will be discussed separately under of the following headings:

1. Support Routines
2. Executive Routine
3. Punctuation Routine
4. Search Routine
5. Rule Macro Routines

The interrelation of the above routines will be explained and then each will be discussed in detail. The transcription routine uses three areas of memory located in RAM. The RAM areas are the workspace, the video RAM buffer and the print buffer. The video RAM buffer is a holding area for text until transcription occurs. The workspace area is an exact copy of the text found in the video RAM buffer. The print buffer is used to hold the transcripted text. The code in the print buffer is specially created to match the characteristics of the printer and has no universal format. The ROM section of the memory system is divided into five parts: program storage, prefix tables, suffix tables, wordsign tables and abbreviation tables. The tables are used

to match text words with known table contents, and thereby specify the appropriate entry points in the rule routines. The rule routines then determine if the match between the text word and the table word is to be accepted or rejected.

Support Routines

The support routines are subroutines that are used to perform specific operations before or after one of the major routines. The following is a description of the more important support routines.

1. Prefix, Suffix, and Vowel Search

This routine searches through the text word being transcribed to determine if a prefix and/or suffix exists(10,11,12).

The first step in determining if the text word has a prefix or suffix is to check the total length of the word. Since a word in English containing a prefix or suffix always has a root word the first step must be to determine the minimum length of the root word in the text word. It is not always possible to know what the root is exactly because a table of all the words in the English language would be required. An empirical rule has been developed to define a root as far as this computer system is concerned. A root will consist of no fewer than three letters one of which must be a vowel.

On entering this routine the total length of the word is already known having been calculated by

the punctuation routine. The minimum root length is assumed and a search begun of the remaining letters, if any, preceeding the assumed root word. A search is made through tables of prefixs contained in ROM. The search is started with the maximum allowable prefix that could be found in the word and if a possible match is fcund, the letters remaining are checked to see if they contain a vowel. If a vowel is found the prefix in the table is assumed to be the prefix of the text word. A control word is set to store the length of the prefix and the fact that a prefix was found. However, if the assumed root does not contain a vowel then the root word length is increased by one and the prefix length decreased by one and the search through the prefix tables continued. This procedure continues until a prefix is found or it is determined no prefix exists. In any case, the results of the search will be stored in a control word for the use of subsequent routines.

Once it is determined whether or not the a prefix exists in the text word, it is necessary to determine if a suffix exists or not. The maximum length of suffix possible is calculated taking into account the length of the prefix, if any. A search is made through the suffix tables for a suffix of this length until either a suffix is found or it is

determined that a suffix does not exist. Should a match be found the root word remaining is seached for a vowel. As in the search for a prefix, the root word must contain a vowel for a suffix to be accepted. If a vowel is not found the suffix is rejected and the suffix search length decreased by one and the root length increased by one. This procedure will continue until a suffix is found or until it is determined a suffix does not exist. The results of the suffix search are stored in a control word.

Upon exit from the 'Prefix, Suffix, and Vowel Routine' two control words will have been set to reflect the presence or abscence of a suffix or prefix. These control words are of prime importance in the routines which implement the rules of Braille because many of the rules of Braille depend upon the presence of prefixs, suffixs, and syllables.

2. Syllable Routine

The term 'wordsign' refers to the Braille representation of an English word. There are two main tables in ROM that are used by the Braille routines; the 'wordsign' table and the 'abbreviation' table. The 'word sign' table contains the code that represents the Braille symbols for all the English words and groups of letters used as contractions in the Braille language. The

'abbreviation table' contains the code that represents the Braille symbols for the English words that are used as abbreviations in the Braille language.

The purpose of the syllable routine is to determine if overlap occurs between a word match in the word sign and/or abbreviation tables and an assumed syllable. Prefixs and suffixs always comprise syllables in words. Some of the rules in Braille are dependant upon knowing if a syllable exists at the start of a word or not. A syllable is assumed to exist at the start of the word if a prefix has been found, and similarly, a syllable is assumed to exist if a suffix has been found. Although this algorithm does not cover all the cases it will been seen that the largest part is covered and those not covered will be dealt with in another way.

The determination of the presence of a syllable, if any, is made using the information present from the prefix and suffix control words. A calculation is made to determine if an overlap occurrs between the syllable and the current match from the 'word sign' table and the results of this decision are returned as a code in the accumulator to the calling routine. The calling routine is free to make a decision on the basis of the results of

the syllable search.

3. Initialize Pointers

This program is a set of routines that when called by a particular program, clear a workspace or set up pointers for future use by the calling program.

Executive Routine

The executive routine performs the task of determining which routines will be called and in what order. Once it is determined that every word has been transcribed in the current text area the executive routine returns control to the calling program which is always the 'Control Routine'.

The executive is entered by the operator typing ESC then T for transcribe. This procedure will signal the control routine that transcription is to occur. Program control is now passed to the executive which prepares for transcription by copying the text into the work space RAM and clearing the current video screen. Control words that are reset at the start of transcription of each page are now reset. A loop is begun to transcribe the entire text and a check is made for the end of the text after transcription of each word. The first routine called is the punctuation routine. Upon return from this routine one text word and its surronding punctuation will be described. The next routine called is the search routine. This routine will find all of the possible contractions and abbreviations in the text word. Whether or not the contraction or abbreviation found

will be an acceptable match will be determined by the various rule routines. This loop is continued until all of the text has been transcribed and the transcribed code has been placed in the print buffer for output to the printer.

Punctuation Routine

Text is transcribed word by word, with the punctuation following the word determining when search and replacement by Braille symbols will actually occur. As each word is encountered in the text, control words are created to define all surrounding punctuation and the number of letters in the word.

When an alphabetic character is found, it is known that no more punctuation will be found preceeding the word and the punctuation preceeding the text word is placed into the print buffer. The text work space continues to be searched until the end of the current word is detected. It is assumed that when a space or succeeding punctuation (period, comma, exclamation point etc.) are encountered that the end of the word has occured. Upon finding a character that signals that the end of the word has occured a call to the search routine is made. The punctuation routine will check that the end of the page has not been reached before calling the search routine. When the end of the page is reached a flag is set by the punctuation routine to notify the executive routine that transcription of the entire text area is complete.

Search Routine

The search routine searchs the current text word for

the various Braille contractions and abbreviations that may make up the word. When a match has been found between a string in the text word and the contraction tables, the rule for that contraction is checked and if the contraction is acceptable it is retained as part of the transcribed word. The letters remaining that have not been contracted will be searched for any additional contractions. This process will continue until there remains no more letters to contract in the current text word or until the search routine verifies that all possibilities are checked in the text word and there exists no further reason to continue searching. Upon exit from this routine, the text word will be coded in its contracted state and awaiting placement in the print buffer. Control is now returned to the executive routine and the text character is transferred to the print buffer.

I/O Routines

The routines described as 'I/O Routines' have not been written due to a lack of program development resources and time for this project however, the general structure of these programs have been given to illustrate the total concept of the project.

The only devices that operate as input/output devices are the keyboard and printer. The keyboard has already been discussed therefore, this discussion will only deal with the printer.

The Braille printer as described earlier consists of three parts; paperfeed, carriage, and the embossing

mechanism. The software for the paperfeed and carriage control are similiar. Each mechanism is driven by a stepper motor and the stepper motor is driven by a linear integrated circuit. These integrated circuits are in turn controlled by output lines on a VIA port.

When the 'I/O Routine' has been entered it is because the print buffer is full and must be output to the printer. It is now the task of software to co-ordinate the movements of the carriage, the paperfeed and the embossing mechanism to create the Braille characters. This task is done using the following five routines:

1. File Management
2. Print Routine
3. Solenoid Routine
4. Carriage Routine
5. Linefeed Routine

Each Braille symbol in the print buffer represents a matrix of six dots and, as has been seen, the printer will emboss a column of three dots at any one time. It is therefore nescessary to emboss two columns to create one Braille symbol; it is not possible to emboss the whole column of the same character due to the physical spacings of the solenoids. Note that more than one Braille character is worked on at one time. The three dots are punched by column embossing every other character until the entire line is done and every dot is accounted for.

File Management

Before printing begins the print buffer must be sorted and arranged in a format that can be easily managed by the CPU. The Braille characters are arranged so that they correspond to the solenoid placement. Lines of Braille are scanned to determine the maximum number of characters that may be placed on the line without breaking the word. A set of pointers and a control word are derived for each line after the line has been scanned. The control word contains the number of characters in the line and bcause the printer is bi-directional a flag indicates the direction of printing. This set of control words make up the Braille file. The file management routine initializes the file parameters for every line that is to be output to the printer. The 'Print Routine' adjusts the file as the Braille characters are output.

Print Routine

Once the print buffer has been assembled it is the task of this routine to control the individual components of the Braille printer and adjust the file parameters as each line is output. When the print routine is first entered, the printer is initialized to the right hand corner of the paper, the column data is sent to the solenoids, the 'Solenoid Delay' routine is called and a return from the interrupt is made to allow the rest of the system to continue processing data. When

the delay time is complete, an interrupt is generated by the VIA timer and a return is made to the I/O routine. The next step is to move the carriage to the next spot where a character is to be embossed. The 'Carriage Activate' routine is called and the carriage is moved to the next cell position either right or left depending upon which direction is being printed. During the time the carriage is being moved, a return from the interrupt is made to allow the CPU to continue processing data. When the timer for the carriage delay interrupts the CPU, return is made to the I/O routine. The I/O routine now checks if more characters remain in this line, and if so, they are printed in the manner above. If more characters remain but are not in this line, the 'File Routine' is called to assemble the next set of parameters for the next line. This process will continue until the print buffer is completely empty.

Solenoid Routine

The purpose of this routine is to output column data to the solenoids and to initiate a time-out (and interrupt) of the timer in the VIA. The interrupt is used to notify the CPU that the settling time of the solenoids is over. After initializing the timer and pulsing the solenoids, this routine returns to the 'Print Routine'.

Carriage and Paperfeed Routines

These two routines are identical in nature and so

will be explained as one. These routines control the stepper motors which move the solenoid block and paper roller. The paper roller is always operated in one direction while the carriage is bi-directional. A linear integrated circuit steps the motor using a trigger pulse from the VIA. As soon as the motor has been activated a timer is initialized to interrupt the CPU after a determined time delay. A return is made to the calling program until the timer interrupt occurs.

IV. Braille Macro Routines

This section is best understood by referring to the description of the Braille rules given in Appendix A. In each case where simplifying assumptions have been made and/or an empirical rule derived to implement a partcular rule of Braille, the reasons for the assumptions will be stated.

It should be noted that the most difficult task in implementing the rules governing Braille is the abstract form of the structure of the English language. Meaning and pronounciation are forms of a language that have cultural, historical, and regional dependance. It is therefore impractical to seek hard and fast rules that will always specify the Braille. The approach used here has been to, use empirical rules to achieve an acceptable rate of correct contraction. This method in some instances, leaves the text uncontracted. However, this approach was deemed best since the meaning of the text is left free of contraction error. The penalty paid is that in a few instances the amount of space needed to write the text is slightly greater.

The next major difficulty in transcribing Braille is the syllabification of words. That is, certain rules in Braille depend upon knowing the prefixs, suffixs, and major and minor syllables of the words. Words are generally hyphenated at the end of a line in order to conserve space and this affects some of the upper and lower contraction Braille rules. One restriction is made at the onset to

reduce the program complexity and increase the number of correct transcriptions; that is, hyphenation at the end of lines is not allowed. This restriction conforms to the method of text entry which does not allow the operator to control the carriage return during text editing. The print routine will place the maximum number of characters per line on the Braille line without breaking the words.

Compound words are another difficult area of transcription. Contractions are not permitted between the component parts of compound words. It is impossible to tell words that are compounded unless a dictionary of the entire English language is stored in memory. This method is obviously not feasible and therefore an alternative approach has been used. The responsiblity of determining the compound words is left to the typist as an option. If the typist so choses, the compound words may be identified visually and using an editing facility, a special non printing character may be inserted at the boundary of the words. This special character, although displayed on the screen, will not be printed in the final output. The character will be detected by the transcription routines however and noted as the boundary of a compound word. Contractions will therefore be inhibited from bridging across the component parts of the compound word. If the typist choses to disregard this option then in the few instances in which a contraction would have bridged the compound word the Braille rule will be contravened.

A. Simple Upper Wordsigns

Simple upper wordsigns are Braille signs that occupy cnly one cell and contain a dot on the top line of the cell. Word signs are contractions that are used to replace entire words. When a simple upper wordsign is detected, a check is made to see and if the length of the match in the tables is equal to the total length of the text word, if it is not this particular match is rejected. A check is made for an apostrophe preceeding the word and if an apostrophe is found the match is also rejected. However if all of these conditions have been met then the match is accepted.

B. Wordsigns

The set of words termed 'wordsigns' must be written unspaced from one another if they occur in the same phrase. It is not possible to determine phrasing of sentences by processing only the separate words of the sentence. Certain combinations of these words can occur but not in the same phrase. The combinations "for a", "of for", and any of the wordsigns followed by the word "and" are usually not in the same phrase. All other combinations of the wordsigns will have the intervening space deleted. Deletion of the space is done in by overlaying a non printng character in the print buffer. Deletion of a space between wordsigns may occur only if the space exists between the wordsigns. Punctuaton and/or other symbols are not allowed between the symbols.

C. Contractions

The words listed under wordsigns may also be used as contractions. A contraction is the replacement of two or more letters by a single cell Braille symbol where the set of letters replaced may be a part of a longer word. The match is first checked by a subroutine to see that a prefix and/or suffix does not overlap the match. If overlap occurs then the match is rejected, otherwise it is accepted. It is a general rule of all contractions that an overlap is not allowed between a prefix or suffix and the contraction. Contractions are not allowed to bridge between the component parts of compound words. The rules governing the use of contractions with prefixs and compound words, as outlined above, have been implemented in all the routines that deal with contractions.

D. Upper Contractions

This set of contractions may be used in any part of a word however, preference should be given to any alternative contraction possible. Overlap between the contraction and a prefix or suffix is not allowed. Since the search algorithm searchs the text word for the longest possible contraction first and then searchs all other possible contractions of shorter length, it is a natural consequence that when any of these contractions are encountered no other contractions are possible. The following is a list of upper contractions.

1. CH,GH,TH,WH : these contractions may be used in any part of the word. The only rule that can not be contravened is the syllabification rule explained earlier.
2. ED, ER, OU, OW : the letters preceeding these contractions are checked to see if they would involve the contraction in a diphthong or diaereses and if so the contraction is rejected.
3. ST : this contraction is dependant on the general rules of transcription, and also on the rule that states that this contraction may not be used in place names ending in 'town', and preceeded by the letter 's'. This rule is easily implemented by checking the text, preceeding and following the contraction, for the letters of interest. The acceptance of this contraction is dependant on the abscense of these letters.
4. AR : this contraction must not be used if its letters fall into a trigraph. The two letters immediately preceeding the contraction are searched to see if they are the letters 'ee', and if so the contraction is rejected.
5. ING, BLE : these contractions may not occur at the beginining of a word. All that need be checked is the position of the contraction within the text word. If the contraction is found to be at the start of the text word then it is rejected.

E. Lower Contractions

Lower contractions are single cell Braille symbols that replace two or more letters in a contracted word. The Braille symbols do not have dots on the top line and hence the name 'lower contraction'. The following is a list of lower contractions.

1. COM : this contraction may be used whenever it occurs at the start of a Braille word. The position of the contraction relative to the entire text word is checked and if the contraction occurs at the start of the word the contraction is accepted.
2. CON, DIS : if either of these contractions form the first syllable of the word they may be used; and if not, they must be rejected. The position of the contraction is first checked to see if it is at the start of the word. If it is the length of the remaining root is caculated. The assumed root word must be greater then three letters, and must contain a vowel. Should all of these conditions be met a further condition is required. A dot 1 or 4 must be present in the word and if a dot exists on the upper line (position 1 or 4) of the Braille cell then he contraction is accepted.
3. BE : the 'be' contraction is treated exactly the same as the preceeding contractions except that an additional empirical rule has been added to determine if this contraction is the first syllable of the word. It can be seen from searching an English dictionary that the

letters 'be' generally comprise the first syllable of a word when the fourth letter in the word is one of the letters 'a, e, i, o, u, h' . Thus the forth letter of the word is checked for one of these letters. This rule does not apply in every instance, for example in the contraction of proper names an error could occur. One instance where where an error occurs is in the name 'BERLIN'. In this example 'be' is not the first syllable of the word.

4. BB, CC,DD, FF, GG : these contractions are known as double letter contractions and are used only between letters and never at the beginning or at the end of a word. Any alternative contraction should be used in preference to these contractions. It is a natural consequence, however of the search algorithm, that should any of these contractions be found there exists no other possiblity of contraction. It is also necessary that a dot exist on the upper line of at least one of the remaining Braille cells for one of these contractions to be used. The contraction is first checked to ascertain that it is surrounded by letters on both sides and if so the processing continues. The contraction is next checked to see if a dot is present on the top line of the Braille word and if so the contraction is accepted.
5. EA : the 'ea' contraction should not be used if any other contraction may be used. This contraction should

not be used with the endings 'al, an, ae'. The contraction must be used between letters and must also have a Braille cell present that contains a dot on the upper line.

The text word is checked to determine if letters exist on each side of the contraction and if so processing continues. The letters following the text word are checked to see if they are one of the previously mentioned endings and if so the contraction is rejected. A check is now made to determine if this contraction is part of the 'eea' trigraph. The letter immediately preceeding the contraction is checked for an 'e' and if the letter is an 'e' the contraction is rejected.

6. EN : this contraction may be used in any part of the word providing it is in contact with dots one or four. The contraction must not form part of a diphthong or diaereses. The letters, if any, immediately preceeding the contraction are checked to see if a diphthong or diaereses exists, if one does not exist, the contraction is accepted.
7. IN : this contraction is used in any part of the word.

F. Lower Wordsigns

Lower wordsigns are single cell Braille symbols that replace entire words. These symbols do not contain either dot one or dot four. All of the following lower wordsigns

must be written without added letters or punctuation; 'be, were, his, was, enough, in'. It will be seen that the wordsigns 'into, to, by' must be written adjoining the wordsign that follows. This restriction is satisfied by deleting the space that separates the Braille symbols. It is now apparent that the deletion of this space will determine whether or not the following wordsigns may or may not be used. Thus the wordsigns 'his, were, be, was, enough, in' will be accepted as valid replacements of English words only if there is a space immediately preceeding them. This is done by looking at the previous character stored in the print buffer. The following is a list of lower wordsigns.

1. BE, WERE, HIS, WAS : these lower wordsigns must stand alone without added letters or punctuation. The punctuation routine checks for the punctuation that surrounds the text word and also keeps track of the number of letters that make up a word. Thus the rules governing these contractions are checked against these parameters and the contraction accepted or rejected accordingly.
2. ENOUGH, IN: these wordsigns may be used in contact with the hyphen, dash, and apostrophe. If they are used with these signs they must be in contact with a dot one or four. This rule is simplified somewhat in that this wordsign is only used if it is not in contact with any punctuation.
3. TO, INTO, BY : these wordsigns must be used adjoining

the Braille sign that follows. However, if the second word that follows is also a lower wordsign then the second wordsign must not be used. It is a general rule of lower signs that they must be in contact with a cell containing a dot one or four. The lower wordsign must not be used if a natural pause occurs after the wordsign. In general, a natural pause is seen to occur if a conjunction follows the wordsign. The wordsign must not be used if punctuation immediately follows or if no word follows it.

The first consideration of the program is to check that only an exact match has been found and if not the wordsign is rejected. Next a check is made of the control words specified by the punctuation routine to see if any punctuation surrounds the wordsign. If punctuation is found, the wordsign is rejected. The word following the present word is now checked to see if one of the conjunctions 'and, or, but' follows and if so, the wordsign is rejected.

The punctuation routine maintains an indirect pointer of the last space stored in the print buffer. This pointer is used to delete spaces between words by storing a special non printing code at that location in the print buffer. When one of the wordsigns 'to, into, by' is found a flag is set to indicate that the next space must be deleted. The space is detected and deleted by the punctuation routine. A check is also made in the print buffer to insure that the wordsigns 'his, was, were, be' will not be in contact with

the lower wordsign in question ('his, was, were, be' must stand alone). If the word to be transcribed satisfies all of the above criteria, it is contracted and the flag set to allow the next space to be deleted. If not, then the wordsign is rejected and transcription continues.

G. Initial Wordsigns

These Braille signs occupy two consecutive positions and may be used to form wordsigns and contractions. Some of the 'initial wordsigns' depend upon meaning and pronounciation. Since it is an impossible task to determine these characteristics using a computer, simplifying assumptions will be made when necessary. These assumptions will be stated and explained as they are encountered for a given word or set of words.

All of the initial wordsigns may be used if they occur without added letters. The count of the number of characters in the text word is checked against the number of letters in the match and if they are equal, the wordsign is accepted. The punctuation routine maintains a count of the number of characters in the text word as one of its parameters. The following is a list of initial wordsigns.

1. DAY, FATHER, KNOW, LORD, MOTHER, QESTION, RIGHT, OUGHT, MANY : these wordsigns may be represented as contractions whenever they occur regardless of where or how they occur.

All of the following intial wordsigns may be used

as contractions if the letters they represent within the word to be contracted are pronounced exactly as they would be if standing alone; 'ever, here, name, one, time, under'.

2. EVER : the wordsign 'ever' may be used as a contraction, and will be contracted correctly in the majority of cases, if the letters 'ever' are preceeded by any two letters and is not followed by any letters. There will be some instances in which the contraction could have been used but was not. However, these incidents will not result in an error in the Braille output, but rather a small amount of extra code will be generated.
3. HERE : the wordsign 'here' may be used as a contraction when the letters it is to contract forms a syllable with the 'h' pronounced. The same rule as was used for the wordsign 'ever' is used for the wordsign 'here'.
4. NAME : whenever these letters are found and are pronounced as the word 'name' they may be replaced by a contraction. The above is the case in almost every instance where the word 'name' occurs, and so no rule was implemented to identify those cases in which there would be an exception.
5. TIME : this word may have a maximum of any two letters preceeding the contraction and any number following. This rule is implemented by checking the postion of the contraction within the text word and calculating the number of letters that preceed and follow the word to be

contracted.

6. SOME : this word is never allowed to replace part of a word but can be used as a complete wordsign.
7. ONE : 'one' may be contracted whenever its letters are pronounced as a single syllable. This rule is implimented by allowing only one letter to preceed the contraction and one letter to follow the contraction. The letter following the contraction is not allowed to be one of the following; 'd, n, r'. If one of these letters follow the assumed contraction the appropriate two letter contraction is used instead.
8. UNDER : any number of characters may follow the word 'under' if used as a contraction except the letters 'o and i'. An example of where the word 'under' may not be used as a contraction, is in the word 'underived'. The letters of the assumed contraction are not pronounced as 'under'. This rule is implemented by calculating the position of the contraction within the word and then determining the number of letters that surround the contraction. The decision to accept or reject the contraction is made according to the above criteria.

The remaining initial wordsigns depend upon meaning for their use as contractions. Again empirical rules have been derived to cover as many of the possible correct contractions as possible. Each rule will be explained as it is encountered. The following is a list of the remaining initial wordsigns.

9. WORK, YOUNG, THERE, CHARACTER, THROUGH, WHERE : these contractions should be used only when their meaning is retained. Since there are extremly few exceptions found in the English dictionary where these words are used as part of another word and their meaning is not retained, they are always contracted by the Braille programs whenever found.
10. UPON, THESE : these words are never allowed as contractions but may be used as wordsigns.
11. WORD : this word may be contracted if no letters preceed the assumed contraction. However, any number of letters may follow the contraction.
12. THOSE, WHOSE, CANNOT, MANY, SPIRIT, WORLD, THEIR : these letters are always contracted whenever they appear as part of a word.
13. HAD : these letters are never contracted if they are part of a longer word.

The following initial wordsign does not depend upon pronounciation or meaning but on the tense of the verb or upon the letter following the assumed contraction.

14. PART : These leters may be used as a contraction whenever the letters occur except when followed by the letter 'h' or when the prefix 'par' is followed by any part of the verb 'take'. These letters may not be used as a contraction in the following examples; 'parthenon, partake'.

H. Final Contractions

These contractions are formed by a double Braille symbol and as the name implies must not be used to represent an entire word but rather part of a word. These contractions may occur at any place in the word except at the begining and they are also not allowed to be in contact with a hyphen. The following is the complete list of final contractions.

1. ANCE, ENCE, LESS, NESS, OUND, OUNT, ONG, MENT, FUL, ITY, ATION, ALLY : the position of the assumed contraction is checked against the parameters created by the punctuation routine as to position. If the contraction is not contained within a larger set of letters or if the assumed contraction occurs at the start of the text word, the contraction is rejected. In all other cases the contraction is accepted as valid.

V. Summary

The operation of the Braille transcription system was verified as the software programs were being developed. The information in the print buffer was checked against known transcripted code for all the forms of the Braille contractions and abbreviations and it was seen that the Braille transcription system operated as designed. However, the programs to drive the Braille printer have not been written. The general structure of these programs have been outlined in the section of the thesis dealing with software. There is currently neither software or hardware support, at this university, for the MCS6502 microprocessor and in order to shorten the length of time spent on this project the printer programs were not developed.

The capabilities of the 'transcriber' can be extended by addition of software. In particular, many 'intelligent terminal' capabilities may be added with little extra software effort. The ability to clear the text by word or line and adjust the remaing text accordingly, may be implemented by using many of the editing subroutines now in existence. The 'transcriber' may be turned into an 'intelligent terminial' for the blind without the use of the video screen. This feature could be implemented by adding a serial communications port in hardware and by adding the software for the 'transcriber' to operate the serial interface. This feature would provide the blind programmer with a hardcopy terminal equivalent to the hardcopy

terminals used by sighted programmers.

The next phase of devlelopment of the 'transcriber' would be to design a production model from the existing prototype. This design would involve the elimination of all the hardware used in the development portion of the microprocessor system. The memory modules of the 'transcriber' would be designed around newer and more dense memory arrays, which in turn reduce the component count and simplify the manufacturing process. The video section of the 'transcriber' would be designed around one of the new CRT cotroller circuits. This feature would eliminate a great deal of complexity in the overall construction of the 'transcriber'. The MCS6502 still remains the best choice of processor for this machine, since at the present time the MCS6502 is still the only microprocessor that offers true indirect indexed addressing.

VI. References

1. Magnuski, H.
   Computer Aids to The Handicapped - The PDP-8 as a Braille Trans lator.
   Project MAC. Massachusetts Institute of Technology
   Decus Proceedings, Spring 1966
2. Pace Technical Description.
   4200078m N.S.C. 1975
3. M6800 Microprocessor Applications Manual.
   Motorola Inc. 1975
4. Introducing the 2650.
   Signetics Corp.
5. Intel 8080 Microcomputer Systems User's Manual.
   Intel Corp. 1975
6. MCS6500 Microcomputer Family Hardware Manual.
   6500-10 1975 Mos Technology Inc. 950 Norristown, PA. 19401
7. MCS6500 Microcomputer Family Programing Manual.
   6500-50 1975 Mos Technology Inc. 950 Norristown, PA. 19401
8. Godbout Inc. 1975.
9. Polymorphics Inc. 1975.
10. Smith, G.L.
    Spelling by Principles.
    Meredith Publishing Company, 1966.
11. Nathan, N.
    The Right Word.

Houghton Mifflin Company, Boston, 1962.

12. Brown, J.I.
Programmed Vocabulary, Second Edition.
Meridith Publishing Company, 1971.

13. Bradley, M., Bromly, M., Gorman,P.
English Braille Grade 2
A Course for Transcribers.
The Canadian National Institute for the Blind. 1967

14. Dorf, M.B., Sharry, E.R.
Instruction Manual for the Blind and Physically Handicapped.
Library of Congress, Washington, D.C. 20542. 1973

VII. APPENDIX A: The Rules of English Braille

The language of Braille is generated by the replacing of English words by wordsigns, contractions and one to one mappings of English and Braille (13,14). The term wordsign refers to an English word that is replaced by a Braille sign and a contraction refers to a set of letters within an English word that is replaced by one or more Braille symbols. Wordsigns and contractions can be catagorized as one of seven different types. Each of the types are listed below and an explanation given of each.

1. Simple Upper Wordsigns
2. Wordsigns
3. Contractions
4. Upper Contractions
5. Lower Contractions
6. Lower Wordsigns
7. Initial Wordsigns
8. Final Contractions

A. Simple Upper Wordsigns

These types of contractions are simple because they are represented by one Braile character (one cell). They are described as 'upper' because there is a dot on the upper line of the cell and are called wordsigns because the contraction is used to represent an entire word. No letters may be added to the word to be transcribed execept after an apostrophe. The word may not be transcribed if an apostrophe

preceeds the word. For example the wordsign 'it' may be contracted in 'it's' but not in ''it'.

The set of simple upper wordsigns are represented by the first letter of the word they contract except for the word 'as' which is represented by the letter z. The following is a complete list of the simple upper wordsigns: but, can, do, every, from, go, have, just, knowledge, like, more, not , people, quite, rather, so, that, us, very, will, it, you, as.

B. Wordsigns

The wordsigns described here are used whenever they occur as an entire word. When two or more of them occur in a row, the space separating them is deleted unless a natural pause occurs. A natural pause is said to occur if a comma separates the wordsigns or if a question mark, period or comma could have been inserted between the wordsigns. A natural pause occurs when the wordsigns are followed by a conjunction.

C. Contractions

There are general rules which govern all types of contractions. These rules are listed below and reference made to them by number when they apply to a specific set of contractions.

1. A contraction may not be used if the contraction overlaps a prefix or suffix.

2. A contraction may not be used if its letters would fall into the diphthong or diaeresis 'ae' or 'oe'.
3. A contraction may not be used if its letters bridge between the component parts of a compound word.
4. The letters of a contraction may not be part of the 'ee' diagraph or the 'eau' trigraph.
5. Preference should be given to the contractions that saves the greatest amount of space.
6. Any alternative one cell contraction should be used in preference to a double letter contraction.
7. The contraction which most nearly maintains the correct form of pronunciation should be used.

Upper Contractions

There are three sets of upper contractions to be considered and each will be considered separately. All the contractions have a dot present on the upper line of the cell. Each contraction replaces two or more letters.

1. CH, GH, SH, TH, WH : these contractions may be used in any part of a word except as noted above.
2. ED, ER, OU, CW : these contractions may be used in any part of the word except as noted in the rules governing contractions. The 'OU' contraction is also used as a wordsign that stands for 'out'.
3. ST, AR, ING, BLE : the 'ST' contraction is not used in place names endig in 'town' and preceeded by the letter 's'. The contraction for 'BLE' is not allowed to immediately follow a prefix.

Lower Contractions

The following sets of contractions do not have a dot on the top line of the Braille cell and are therefore called lower contractions. The general set of rules that govern contractions also apply to these contractions.

1. BE, CON, DIS, COM : these lower contractions may be used only when they form the first syllable of a word. However, the contraction 'COM' need not form a syllable, but must occur at the start of the word. In addition to the above, the contraction for 'COM' may never be used with the dash or hyphen.
2. EA, BB, CC, DD, FF, GG : these contractions excluding the 'EA' contraction are known as the double letter contractions, and may be used only between letters and never at the beginning or end of a word. In additon to the general rules of contraction previously listed, the following rule also applies to the 'EA' contraction. In words ending with the suffixs 'able', 'age', 'ate' , 'al',and 'an' the 'EA' contraction is not to be used. Any alternative contraction should be used.

Lower Wordsigns

Lower wordsigns as the name implies are Braille symbols that stand for entire words. As well they do not have any dots on the upper line of the Braille cell. These wordsigns must be written without added letters or punctuation because of the fact that the symbols used to represent these words also double as symbols for some of the punctuation symbols

in Braille.

In general any number of lower wordsigns may be written unspaced from one another providing they are in contact with a Braille symbol containing either a dot one or four. If, however, the dot one or four is not present the last lower wordsign contraction in the series must be omitted. The following is a list of lower wordsigns.

1. BE, WERE, HIS, WAS : when these words occur with added letters or punctuation they must be written out.
2. ENOUGH, IN : these wordsigns must be spaced from all other Braille signs except the dash, hyphen, and apostrophe. Should they be used with these signs they must be in contact with either a dot one or four.
3. TO, INTO, BY : these wordsigns may be used only adjoining the sign that follows. However, they may not be joined to the sign that follows if the following word is a conjunction such as 'but', 'and', 'or'. These contractions may not be used when: no word follows, punctuation follows, if they occur at the end of a line, or when joined to another word by a hyphen. In addition to the above, these wordsigns may never be joined to the wordsigns 'BE', 'HIS', and 'ENOUGH' since these signs must be written unspaced from all other letters and two lower wordsigns must not be written together without an adjoining dot.

<u>Compound Signs</u>

These types of signs occupy two Braille cells to form

wordsigns and contractions.

1. DAY, FATHER, KNOW, LORD, MOTHER : these contractions may be used whenever they occur.
2. EVER, HERE : 'ever' may be used only when the letters it represents are pronounced as the word 'ever'. The contraction for 'here' is allowed only 'when a syllable of the word contains the contraction with the letter 'h' and is pronounced as the 'h' sound.

Initial Wordsigns

The following contractions are double cell contractions and are generally used to replace entire words although in some cases they may be used as a contraction for part of a word.

1. QUESTION, RIGHT, OUGHT, MANY : these contractions may be used whenever the letters they represent occur.
2. SOME : this contraction may be used when the letters it represents forms a definite syllable of the original word.
3. TIME, UNDER, NAME : these contractions are allowed only when the letters they ae to replace are pronounced exactly as the free standing word used for the contraction.
4. ONE : this contraction may be used if its letters are pronounced as a single syllable.
5. PART : if the letters this contraction is to replace is not followed by the letter 'h' or if the prefix 'par' is not followed by any part of the verb 'to take' then this

contraction may be used.

6. WORK, YOUNG, THERE, CHARACTER, THROUGH, WHERE, UPON, WORD, THESE, THOSE, WHOSE, CANNOT, HAD, SPIRIT, WORLD, THEIR : these contractions may be used only when their meaning is retained.

Final Contractions

Final contractions are double cell contractions, and are used to contract any part of a word except the beginning of the word. These contractions may not be used to form a whole word.

1. ANCE, ENCE, LESS, NESS, SION, TION, OUND, OUNT, ONG, MENT, FUL, ITY, ATION, ALLY : these final letter contractions may not be used after an appostrophe.

VIII. APPENDIX B: MCS6502 PROGRAMMING ARCHITECTURE

INSTRUCTION SET – OP CODES, Execution Time, Memory Requirements

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND,X)			(IND),Y			Z,PAGE,X			ABS,X			ABS,Y			RELATIVE			INDIRECT			Z,PAGE,Y			CONDITION CODES					
MNEMONIC	OPERATION	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
ADC	A+M+C → A (1) (4)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										✓	✓	✓	-	-	✓
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										✓	✓	-	-	-	-
ASL	C ← 7 ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													✓	✓	✓	-	-	-
BCC	BRANCH ON C=0 (2)																															90	2	2							-	-	-	-	-	-
BCS	BRANCH ON C=1 (2)																															B0	2	2							-	-	-	-	-	-
BEQ	BRANCH ON Z=1 (2)																															F0	2	2							-	-	-	-	-	-
BIT	A∧M				2C	4	3	24	3	2																															M_7	✓	-	-	-	M_6
BMI	BRANCH ON N=1 (2)																															30	2	2							-	-	-	-	-	-
BNE	BRANCH ON Z=0 (2)																															D0	2	2							-	-	-	-	-	-
BPL	BRANCH ON N=0 (2)																															10	2	2							-	-	-	-	-	-
BRK	(See Fig. 1)													00	7	1																									-	-	-	1	-	-
BVC	BRANCH ON V=0 (2)																															50	2	2							-	-	-	-	-	-
BVS	BRANCH ON V=1 (2)																															70	2	2							-	-	-	-	-	-
CLC	0 → C													18	2	1																									-	-	0	-	-	-
CLD	0 → D													D8	2	1																									-	-	-	-	0	-
CLI	0 → I													58	2	1																									-	-	-	0	-	-
CLV	0 → V													B8	2	1																									-	-	-	-	-	0
CMP	A - M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										✓	✓	✓	-	-	-
CPX	X - M	E0	2	2	EC	4	3	E4	3	2																															✓	✓	✓	-	-	-
CPY	Y - M	C0	2	2	CC	4	3	C4	3	2																															✓	✓	✓	-	-	-
DEC	M - 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													✓	✓	-	-	-	-
DEX	X - 1 → X													CA																											✓	✓	-	-	-	-
DEY	Y - 1 → Y													88																											✓	✓	-	-	-	-
EOR	A ⊻ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										✓	✓	-	-	-	-
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													✓	✓	-	-	-	-
INX	X + 1 → X													E8	2	1																									✓	✓	-	-	-	-
INY	Y + 1 → Y													C8	2	1																									✓	✓	-	-	-	-
JMP	JUMP TO NEW LOC				4C	3	3																												6C	5	3				-	-	-	-	-	-
JSR	(See Fig. 2) JUMP SUB				20	6	3																																		-	-	-	-	-	-
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										✓	✓	-	-	-	-

MNEMONIC	OPERATION	IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND,X)			(IND),Y			Z,PAGE,X			ABS,X			ABS,Y			RELATIVE			INDIRECT			Z,PAGE,Y			CONDITION CODES					
		OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3										✓	✓	-	-	-	-
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2																															✓	✓	-	-	-	-
LSR	0 → 7 → 0 → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0	✓	✓	-	-	-
NOP	NO OPERATION													EA	2	1																									-	-	-	-	-	-
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										✓	✓	-	-	-	-
PHA	A → M_S S-1 → S													48	3	1																									-	-	-	-	-	-
PHP	P → M_S S-1 → S													08	3	1																									-	-	-	-	-	-
PLA	S+1 → S M_S → A													68	4	1																									✓	✓	-	-	-	-
PLP	S+1 → S M_S → P													28	4	1																									(RESTORED)					
ROL	C ← 7 ← 0 ← C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													✓	✓	✓	-	-	-
ROR	C → 7 → 0 → C				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													✓	✓	✓	-	-	-
RTI	(See Fig. 1) RTRN INT													40	6	1																									(RESTORED)					
RTS	(See Fig. 2) RTRN SUB													60	6	1																									-	-	-	-	-	-
SBC	A - M - C → A (1) (4)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										✓	✓	(3)	-	-	✓
SEC	1 → C													38	2	1																									-	-	1	-	-	-
SED	1 → D													F8	2	1																									-	-	-	-	1	-
SEI	1 → I													78	2	1																									-	-	-	1	-	-
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										-	-	-	-	-	-
STX	X → M				8E	4	3	86	3	2																																	-	-	-	-	-	-
STY	Y → M				8C	4	3	84	3	2													94	4	2																	-	-	-	-	-	-
TAX	A → X													AA	2	1																									✓	✓	-	-	-	-
TAY	A → Y													A8	2	1																									✓	✓	-	-	-	-
TSX	S → X													BA	2	1																									✓	✓	-	-	-	-
TXA	X → A													8A	2	1																									✓	✓	-	-	-	-
TXS	X → S													9A	2	1																									-	-	-	-	-	-
TYA	Y → A													98	2	1																									✓	✓	-	-	-	-

(1) ADD 1 TO 'N' IF PAGE BOUNDRY IS CROSSED

(2) ADD 1 TO 'N' IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO 'N' IF BRANCH OCCURS TO DIFFERENT PAGE

(3) CARRY NOT BORROW

(4) IF IN DECIMAL MODE Z FLAG IS INVALID.
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_S MEMORY PER STACK POINTER

ADD
SUBTRACT
∧ AND
∨ OR
⊻ EXCLUSIVE OR
✓ MODIFIED
- NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6

N NO CYCLES
NO BYTES

IX. APPENDIX C: Edit Commands

The following commands are used to edit and control the format of the text on the display. All of the commands must be entered in the control mode and may be used with the repeat key if multiple enteries of the same command are required. Movement of the cursor will not affect the contents of the display in any way if the only function to be performed is the relocating of the cursor.

1. A- this command causes the blinking cursor to move forward by one character position.
2. B- this command backspaces the cursor one character position.
3. C- this command clears the entire line and the cursor returns to the start of the same line.
4. E- this command backspaces the cursor one line and will wrap the cursor around if the top of the page is exceeded.
5. F- this command linefeeds the cursor by one line and will wrap the cursor around from top to bottom if the page is exceeded.
6. H this command homes the cursor to the upper left-hand corner of the display.
7. K- this command clears the line of text to the right of the cursor. The cursor remains in the same position.
8. LINEFEED- this command advances the cursor by eight lines and will wrap the cursor around if the bottom of the page is exceeded.

9. M or CARRIAGE RETURN- this command returns the cursor to the start of the present line; the text is unaffected.
10. N- this comand advances the cursor 16 spaces forward, the cursor will wrap around to the same line. This feature is used when several spaces are needed to be moved.
11. TAB- this command returns the cursor to the start of the next line.
12. O- this command is used as a recovery from the following edit mode.

    The following commands are also in the edit mode but since they are more significant in their consequence they must be preceeded by the ESC key. The control mode is not used with these functions. The blinking cursor will not be displayed until the command is completed or until recovery is made from this mode by using control 'O'.

13. F- this command causes the previous page on the display to be recalled and displayed, the current page is stored in memory and may be recalled by using the control F function a second time. However, if the system is told to transcribe the text on the screen, the previous page is lost and the text on the display becomes the stored page.
14. P- this command clears the entire page and the cursor is homed to the upper left-hand corner.
15. Z- this command clears the display from the cursor to

the end of the page and the cursor position is left unchanged.

X. APPENDIX D: PROGRAM MEMORY MAP

```
B000-B047   FBMAN   (S.R)
B04A-B083   CHGTXT  (S.R)
B084-B09B   FPRTBF  (S.R)
B0B0-B0BE   INTXTB  (S.R)
B0C0-B0CC   PNCSUC  (S.R)
B0C6-B0CC   TRFB    (S.R)
B0CD-B0D9   DECPNT  (S.R)
B0DD-B112   PRECH   (S.R)
B114-B11D   CHKNXT  (S.R)
B11E-B129   PNCPRE
B12C-B129   SORT    (S.R)
B1AC-B1C8   CHAR    (S.R)
B1DO-B1E9   STRPRT  (S.R)
B1ED-B1FB   STRWD   (S.R)
B1FE-B213   MAP     (S.R)
B216-B21E   DSUC    (S.R)
B220-B227   DPRE    (S.R)
B229-B236   CKNUM   (S.R)
B240-B252   WRITE   (S.R)
B253-B269   INPRT   (S.R)
B25B-B269   INVDO   (S.R)
B26A-B298   COPY    (S.R)
B299-B2A5   CLEAR   (S.R)
B2A8-B2B8   SETP    (S.R)
B2BA-B2C0   ADJC4   (S.R)
B2C2-B40A   SEARCH  (S.R)
B410-B422   DASHP   PATCH
B426-B441   WSPARM  (S.R)
B447-B788   PCN     (S.R)
B7A0-B826   PREFIX  (S.R)
B829-B885   VOWEL   (S.R)
B88C-B921   SUFFIX  (S.R)
B928-B963   SYLLAB  (S.R)
B965-B9EC   WWC
BA00-BA1C   CURSOR  INTER
BA1E-BA26   KCHAR   INTER
BA2B-BA7C   KYDEC
BA80-BAA2   FORSP
BAA5-BAC0   BKSP
BAC3-BAA8   CLL
BAD9-BAEE   RVLF
BAF2-BB07   ADVLF
BB08-BB10   HOME
BB14-BB25   CLEOL
BB27-BB2F   HBOL
BB30-BB3D   I/P
```

```
BB43-BB58   RT10
BB5E-BB66   FOR8
BB67-BB6D   DECODE PATCH
BB6E-BB74   OUT
BB75-BBD5   ESC
BBDF-BC3A   INSERT (S.R)
BC3D-BC44   EXIT2
BC48-BC77   DEL'   (S.R)
BC78-BC9C   CLRPGE
BCA0-BCC5   ESC.Z
BCDA-BD06   EXECUT MAIN
BD38-BD58   SUW'S
BD60-BDBE   CNW    (S.R)
BDC0-BDF5   DIPDIR (S.R)
BE00-B2B    INTDEC
BE80-BEA1   UPCNED
BEA5-BEC2   UPCNST
BEC8-BEF2   DBLCHQ (S.R)
BF00-BF46   INIT
C000-C019   UPCNAR
C020-C03C   UPCING
C040-C061   DBCKF  (S.R)
C070-C0CB   LCONCQ (S.R)
C0E0-C113   DOT14  (S.R)
C118-C129   LCNCOM
C130-C17C   LCNCON
C180-C1FA   LCVWL  (S.R)
C210-C22B   BESRCH (S.R)
C238-C300   T2     (S.R)
C310-C32F   LCNBB
C35A-C387   ENGH
C390-C3B0   INT2   (S.R)
C3C0-C3D7   SVTXT  (S.R)
C3D8-C3E4   CHGPNT (S.R)
C400-C40C   UPCNCH
C410-C41B   WWCON
C420-C438   IN
C440-C448   BE
C450-C4C6   EA
C4D0-C4D8   PCNTAB TABLE
C4E0-C4E8   WRDLNG TABLE
C500-C52A   PCNCK  (S.R)
C530-C56B   SRCHST (S.R)
C5E0-C64A   L14
C650-C661   L20
C670-C687   L15
C690-C6FE   L16
C700-C718   PSYLLA PATCH
C720-C730   PSPACE PATCH
C800-C827   L16L18
C900-C910   RESTRT (S.R)
C90C-C91D   XIN    (S.R)
C913-C91D   RESET  (S.R)
C91E-CA33   NUM
```

```
CA3F-CA7A   CHECK   (S.R)
CA80-CA91   LAST    (S.R)
CAA0-CAE4   NUM%
CB00-CB13   PAST    (S.R)
CB20-CB27   APOSTP  PATCH
CB30-CB3A   P$      PATCH
CB40-CB71   INSERT
CB80-CBA2   KYBRDP  PATCH
CBB0-CBC7   DEL
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

XI. APPENDIX E: SOURCE LISTINGS

A. INITIALIZATION ROUTINES

SYSTEM INITIALIZATION

```
THIS PROGRAM INITIALIZES THE VIA PORTS
AND ENABLES THE KEYBOARD AND TIMER
INTERUPTS
INIT    LDA #80          ;SET CURSOR FLAG ON
        STA CURSF
        STA DDRB         ;SET PB7 FOR O/P
        STA ORB          ;SET PB7 HIGH
        LDA #E3          ;ENABLE PORTS, ENABLE
                         ;COUNTER 2 FOR PULSE MODE
        STA ACR          ;STORE CODE IN AUXILARY
                         ;CONTROL REGISTER
        LDA #01
        STA PCR          ;SET CA1 TO INTERUPT ON
                         ;CA1 + TRANSITION
        LDA #7F
        STA IEF#1        ;DISABLE ALL INTERUPTS
        LDA #00
        STA CURSL        ;SET POINTER FOR CURSOR
        LDA #10
        STA CURSH
        LDA #A0
        STA CHAR
        LDA #01
        STA T1L          ;INITIALIZE TIMER 1
        STA T1H
        LDA #A2          ;ENABLE INTERUPT ON CA1
                         ;AND TIME OUT OF TIMER 2
        STA IER#1
        LDA #05          ;INITIALIZE TIMER 2
        STA T2L
        LDA #00
        STA T2H
        STA CNTRL4       ;INITIALIZE TRANSCRIPTION
                         ;PARAMETERS
BKI1    CLI              ;ENABLE PROCESSOR
                         ;INTERUPTS
        JMP BKI1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

INITIALIZE TEXT BUFFER

ALL LOCATIONS IN THE TEXT BUFFER ARE SET TO ALL ONE'S

```
INTXTB   LDA #FF
         LDX #0F          ;SET 16 LOCATIONS TO FF
CONT     STA 60,X
         DEX
         BPL CONT
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

INITIALIZE PRINT BUFFER AND VIDEO POINTERS

```
INPRT INITIALIZES THE PRINT BUFFER POINTERS WHILE
INVDO INITIALIZES THE VIDEC POINTERS
INPRT    LDA #83
         STA PBFH
         LDA #FF
         STA PBL
INVDO    LDA #7F
         STA CPNTH
         LDA #FF
         STA CPNTL        ;SETS POINTER FOR COPY AREA
         STA VPNTL        ;SET POINTER FOR VIDEO DISPLAY
         LDA #0F
         STA VPNTL
         RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

B. INTERUPT DRIVEN ROUTINES

INTERUPT DECODING

```
THE SOURCE OF THE INTERUPT IS FOUND AND THE
PROGRAM DIRECTED TO THE CORRECT
ROUTINE
INTDEC   PHA              ;SAVE ALL REGISTERS
         TYA
         PHA
         TXA
         PHA
         LDA IFR#1        ;FIND THE SOURCE OF
                          ;THE INTERUPT
         AND #02
         BNE KEY
         STA IFR#1        ;TURN OFF INTERUPTS
         JMP CRSR
KEY      LDY #00
         LDA CHAR
         STA (CURS),Y     ;STORE THE CHARACTER
                          ;ON THE SCREEN
         LDA #A0
```

```
          STA CHAR
          JMP KYBRD
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CURSOR

```
THIS ROUTINE IS INTERUPT DRIVEN AND BLINKS A BLOCK
CURSOR ON AND OFF AT A PREDETERMINED RATE
CURSOR    LDA CURSF       ;GET FLAG
          BPL CHARC
          LDY #00
          LDA #09         ;CURSOR CHARACTER
          STA (CURS),Y    ;STORE THE CURSOR
          STY CURSF       ;CLEAR FLAG
OUTC1     LDA #03         ;INITIAL TIMER 2 AGAIN
          STA TM2L
          STY TM2H
          PLA             ;RESTORE REGISTERS
          TAX
          PLA
          TAY
          RTI             ;EXIT FROM INTERUPT
CHARC     LDA CHRC        ;GET CHAR
          STA (CURSOR),Y  ;SET FOR CHARACTER
          LDA #80
          STA CURSF       ;SET FLAG
          BMI OUT         ;BRANCH ALWAYS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

C. TEXT EDITOR ROUTINES

CURSOR ALTERNATE RCUTINE

```
THIS ROUTINE ALTERNATES THE CHARACTER IN
LOCATION CHAR WITH A CURSOR AT THE
RATE DETERMINED BY THE TIMER INTERUPTS
CRSR      LDA CURSF       ;GET FLAG
          BPL CHARC
          LDY #00
          LDA #09         ;GET CURSOR CHARACTER
          STA (CURS),Y
          STY CURSF       ;CLEAR CURSOR FLAG
CUTC      LDA #03         ;INITIALIZE TIMER 2 AGAIN
          STA T2H
          STY T2L
          PLA             ;RESTORE REGISTERS EXIT
          TAX
          PLA
          TAY
```

```
        PLA
        RTI             ;RETURN FROM INTERUPT
CHARC   LDA CHAR        ;GET CHAR
        STA (CURS),Y    ;SET THAT POSITION EQUAL
                        ;TO CHAR
        LDA #80         ;SET CURSOR FLAG
        STA CURSF
        BMI OUTC        ;BRANCH ALWAYS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

KEYBOARD DECODE

THE CHARACTER INPUT FROM THE KEYBOARD IS DECODED AND
THE PROGRAM DIRECTED TO THE NEXT
FUNCTION TO BE PERFORMED

```
KYBRD   LDA ORA#1       ;GET CHARACTER
        CMP #09         ;SEE IF CARRIAGE RETURN
                        ;LINEFEED
        BEQ TAB
        CMP #00         ;SEE IF DELIMIT CHARACTET
        BNE CONTK1
        JMP INSERT      ;PLACE CHARACTER INTO PAGE
CONTK1  JMP DECODE
TAB     LDA CURSL
        AND #C0
        STA CURSL
        JMP ADVLF
ESC1    JMP ESC         ;OUT OF RANGE BRANCH
DECODE  CMP #1B         ;IS IT AN ESC CHARACTER
        BEQ ESC1
        AND #70         ;CHECK FOR A CONTROL WORD
        BNE ESC1
        LDA ORA#1       ;ITS A CONTROL CHARACTER
                        ;GET NEXT INPUT
A       CMP #01         ;FORWARD SPACE?
        BNE B
        JMP FORSP
B       CMP #02         ;BACK SPACE?
        BNE C
        JMP BKSP
C       CMP #06         ;CLEAR LINE?
        BNE E
        JMP CLL
E       CMP #05         ;REVERSE LINEFEED?
        BNE F
        JMP RVLF
F       CMP #06         ;FORWARD LINEFEED?
        BNE H
        JMP ADVLF
H       CMP #08         ;HOME CURSOR?
        BNE K
        JMP HOME
```

```
K       CMP #0B         ;CLEAR TO THE END OF LINE?
        BNE CR
        JMP CLEOL
CR      CMP #0D         ;CARRIAGE RETURN?
        BNE N
        JMP HBOL
N       CMP #0E         ;CURSOR +16?
        BNE LF
        JMP RT10
LF      CMP #0A         ;LINEFEED +8?
        BNE OUTK        ;IF NOT THIS ONE THEN EXIT
        JMP FOR8
OUTK    PLA             ;RESTORE REGISTERS
        TAY
        PLA
        TAX
        PLA
        RTI             ;EXIT FROM INTERUPT ROUTINE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FORWARD SPACE CURSOR

```
FORSP   LDA CURSH       ;FORWARD SPACE CURSOR
                        ;ONE POSITION
        CMP #13
        BNE ADJF
        LDA CURSL
        CMP #FF
        BEQ RTRNF       ;EXIT OUT OF BOUNDS
ADJF    LDA CURSL
        CLC             ;CURSOR+1
        ADC #01
        STA CURSL
        LDA CURSH
        ADC #00
        STA CURSH
EXIT    LDA (CURS),Y
        STA CHAR        ;MOVE NEXT CHAR INTO CURSOR
                        ;POSITION
RTRNF   PLA             ;RESTORE REGISTERS AND EXIT
        TAX
        PLA
        TAY
        PLA
        RTI
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

BACK SPACE CURSOR

```
BKSP    LDA CURSH       ;BACKSPACE CURSOR ONE POSITION
```

```
         CMP #10
         BNE ADJB
         LDA CURSL
         CMP #00
         BEQ RTNBF
ADJB     SEC              ;CURSOR-1
         LDA CURSL
         STA CURSL
         LDA CURSH
         SBC #00
         STA CURSH
RTNB     JMP EXIT         ;EXIT FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CLEAR ENTIRE LINE

```
CLL      LDA CURSL        ;CLEAR THE LINE AND HOME TO
                          ;BEGINING OF LINE
         AND #C0
         STA CURSL
         LDY #3F          ;LINE LENGTH
         LDA #A0
STORE    STA (CURS),Y
         DEY
         BPL STORE
         JMP EXIT         ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

REVERSE LINE FEED

```
RVLF     LDA CURSL        ;REVERSE LINEFEED CURSOR
         SEC
         SBC #40          ;CURSOR-40
         STA CURSL
         LDA CURSH
         SBC #00
         CMP #0F          ;CHECK FOR WRAP AROUND
         BNE OVRT1
         LDA #13
         STA CURSH
         JMP EXIT         ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ADVANCE LINE FEED

```
ADVLF    LDA CURSL        ;ADVANCE LINEFEED CURSOR
                          ;ONE LINE
         CLC
```

```
        ADC #40
        STA CURSL
        LDA CURSH
        ADC #00
        CMP #14          ;CHECK BOUNDS
        BNE OVRT2
        LDA #10
OVRT2   STA CURSH
        JMP EXIT         ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

HOME CURSOR

```
HOME    LDA #10          ;RESET CURSOR TO TOP
                         ;LEFT CORNER
        STA CURSH
        STY CURSL        ;Y WAS RESET ON ENTRY
                         ;TO INTERUPT
        JMP EXIT         ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CLEAR TO THE END OF LINE

```
CLEOL   LDA CURSL        ;CLEAR TO THE END OF THE LINE
        ORA #3F          ;CALCULATE THE NUMBER OF
                          CHARACTERS TO THE END
        SEC
        SBC CURSL
        TAY
        LDA #20          ;PUT A SPACE IN ACCUMULATOR
BKT1    STA (CURS),Y     ;STORE A SPACE
        DEY
        BNE BKT1
        JMP EXIT         ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

HOME TO BEGINING OF TEXT

```
HBOL    LDA CURSL        ;HOME CURSOR TO START OF
                         ;CURRENT LINE
        AND #C0
        STA CURSL
        JMP EXIT         ;RETURN FROM
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CURSOR MOVED RIGHT BY 16

```
RT10    LDA CURSL       ;MOVE FORWARD BY 16
        CLC
        ADC #10
        STA CURSL
        LDA CURSH
        ADC #00
        CMP #14         ;BOUNDS CHECK
        BNE OVRT2
        LDA #10
OVRT2   STA CURSH
        JMP EXIT        ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FORWARD LINEFEED BY 8 LINES

```
FOR8    LDA #02         ;CURSOR+8
        EOR CURSH
        STA CURSH
        JMP EXIT        ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ESCAPE DECODING

```
ESC     LDA #80         ;NOW BEGIN THE DECODING OF THE
                        ;ESC FCN
        ORA ORA#1       ;GET CHAR
        CMP #ESC
        BEQ ESCT
        STA (CURS),Y    ;PUT ON DISPLAY
        JMP FORSP       ;ADJUST CURSOR
ESCT    LDA IFR         ;GET FLAGS FROM INTERUPT
                        ;REGISTER
        AND #02         ;WAIT FOR KEYBOARD INTERUPT
        BEQ ESCT
        LDA #20         ;TURN ON B5 FOR
                        ;UPPER/LOWER CASE
        CRA ORA#1
        CMP #66         ;F?
        BNE I
F       JSR COPY        ;EXECUTE COPY FCN
        JMP EXIT2
I       CMP #69         ;INSERT?
        BNE D
        JMP INSERT
D       CMP #64         ;DELETE?
        BNE P
        JMP DELETE
```

```
P       CMP #70         ;CLEAR PAGE?
        BNE Z
        JMP PAGE
Z       CMP #7A         ;CLEAR TO END OF PAGE?
        BNE T
        JMP ENDPGE
T       CMP #74         ;TRANSCRIBE?
        BNE OVRT3
        JMP EXEC
        JMP EXIT2       ;INPUT INVALID
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

INSERT

```
A CHARACTER IS INSERTED AT THE PRESENT CHARACTER
POSITION AND ALL SUCCEEDING
TEXT IS MOVED FORWARD ONE POSITION, DELIMIT CHARACTER
ALSO ENTERS HERE
INSERT  CMP #00         ;A HAS CURRENT I/P
        BNE LOOPI
        LDA #1D         ;DELIMIT CHARACTER
        JSR TXT+1       ;STORE DELIMIT SYMBOL AFTER
                        ;FORWARD SHIFT OF TEXT
        JMP EXIT2
LOOP    LDA IRF
        AND #02
        BEQ LOOPI
        LDA #80         ;GET I/P
        ORA ORA#1
        CMP #8F
        BNE OVRT4
        JMP EXIT2
        JSR TXT+1
        JMP LOOPI
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TEXT+1

```
SHIFT ALL TEXT FROM CURSOR POSITION AND TO END OF
PAGE FORWARD BY +1 THEN STORE
THE ACCUM AT CURSOR, MOVE CURSOR TO CURSOR+1
TXT+1   STA TMP6        ;SAVE ACCUM
        LDA #00         ;SET POINTERS TO END OF PAGE
        LDA #FF
        STA TRAILL
        STA LEADL
        LDA #13
        STA TRAILH
        STA LEADH
BKT2    LDA CURSH       ;DO MOVE OF TEXT
```

```
          CMP TRAILH
          BNE AD
          LDA TRAILL
          CMP CURSL
          BEQ DONE
AD        LDA LEADL
          SEC
          SBC #01          ;LEAD-1 INTO TRAIL
          STA TRAILL
          LDA LEADH
          SBC #00
          STA TRAILH
          LDA (TRAIL),Y    ;GET NEXT CHARACTER
          STA (LEAD),Y
          LDA TRAILL
          STA LEADL
          LDA TRAILH
          STA LEADH
          JMP BKT2
          LDA TEMP6        ;GET I/P
          STA (CURS),Y     ;STORE ON SCREEN
          STA CHAR
          JSR CURSF        ;CURSOR+1 INTO CURSOR
          RTS
EXIT2     LDA #82
          STA IEF          ;TURN ON KEY BOARD INTERUPT
          JMP EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DELETE TEXT

```
DELETE    JSR DEL          ;CARACTERS ARE DELETED AND
                           ;TEXT IS MOVED TO COMPENSATE
BKT3      LDA IRF          ;WAIT FOR A CHARACTER
          AND #02
          BEQ BKT3
          LDA #20
          ORA ORA#1        ;GET A CHAR TURN B5 ON
          CMP #64          ;EXIT IF ANY OTHER CHAR
                           ;THAN "D" IS RECEIVED
          BEQ BK
          JMP EXIT2        ;ALL DONE RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DELETE CHARACTER AND ADJUST

THE CHARACTER AT THE CURSOR POSITION IS DELETED
AND ALL THE TEXT SUCCEEDING IS
MOVED IN REVERSE BY ONE POSITION THE CURSOR
POSITION REMAINS UNCHANGED

```
DEL     LDA CURSL       ;INITIALIZE THE POINTERS
        STA TRAILL
        STA LEADL
        LDA CURSH
        STA TRAILH
        STA LEADH
BKT4    CLC
        LDA LEADL       ;LEAD+1 INTO LEAD
        ADC #01
        STA LEADL
        LDA LEADH
        ADC #00
        STA LEADH
        CMP #14         ;CHECK LIMITS OF TEXT
        BNE OVRT3
        RTS
OVRT3   LDA (LEAD),Y    ;GET CHARACTER
        STA (TRAIL),Y
        LDA LEADL
        STA TRAILL
        LDA LEADH
        STA TRAILH
        JMP BK4         ;ADJUST POINTER BY +1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CLEAR THE ENTIRE PAGE

```
PAGE    LDA #00         ;CLEAR ENTIRE PAGE
        STA LEADL       ;SET POINTERS
        LDA #10
        STA LEADH
BKT5    LDA #A0
        STA (LEAD),Y    ;STORE SPACE WITH B7 ON
        CLC
        LDA LEADL       ;LEAD+1 INTO LEAD
        ADC #01
        STA LEADL
        LDA LEADH
        ADC #00
        STA LEADH
        CMP #14
        BNE BKT5
        LDA #82
        STA IER
        JMP HOME        ;HOME CURSOR THEN EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CLEAR TO THE END OF THE PAGE

```
ENDPGE  LDA CURSL       ;CLEAR FROM CURSOR TO END
```

```
                        ;OF PAGE
        STA LEADL       ;SET POINTERS
        LDA CURSH
        STA LEADH
BKT6    LDA #A0         ;CLEAR CHARACTER
        STA (LEAD),Y    ;STORE SPACE
        CLC
        LDA LEADL
        ADC #01
        STA LEADL
        LDA LEADH
        ADC #00
        STA LEADH
        CMP #14         ;BOUNDS CHECK
        BNE BKT6
        LDA #A0
        STA CHAR
        JMP EXIT2       ;RETURN FROM INTERUPT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CLEAR DISPLAY

```
COPIES VIDEO INTO A BUFFER AND CLEARS VIDEO DISPLAY
CLEAR   JSR COPY
        LDA #AO
        LDX #00
        LDY #10
        JSR WRITE
        RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CURSOR FEED

```
CURSF   LDA CURSH       ;MOVE CURSOR +1
        CMP #13         ;BOUNDS CHECK
        BNE ADJT
        LDA CURL
        CMP #FF         ;LOWER LIMIT CHECK
ADJT    LDA CURSL
        CLC
        ADC #01
        STA CURSL
        ADC #00
        STA CURSH
        LDA (CURS),Y
        STA CHAR
RTRNF   RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

D. BRAILLE EXECUTIVE ROUTINES

SYSTEM CONTROL ROUTINE

```
THIS POGRAMS CONTROLS AND DIRECTS THE TRANSCRIPTION
CF THE TEXT
EXEC     JSR CLEAR        ;THE SCREEN IS COPIED TO
                          ;THE TEXT AREA THEN CLEARED
         JSR INPRT        ;THE PRINT BUFFER IS
                          ;INITIALIZED
         LDA #00          ;SET FLAG FOR LAST WORD IN
                          ;THE PAGE
         STA FLAGL
         STA CNTRL4       ;INITIALIZE CONTROL WORD FOUR
BKEX1    JSR INTXTB       ;INITIALIZE THE TEXT BUFFER
         LDA #00
         STA PCNTRL       ;INITIALIZE TEXT WORD
                           PARAMETERS
         STA SCNTRL
         JSR PCN          ;BEGIN TRANSCRIPTION
         LDA FLAGL
         CMP #01          ;FLAGL=01 IF PAGE COMPLETE
         BEQ OVREX1
         JMP BKEX1
OVREX1   JSR TRFB         ;FINISH LAST WORD
         LDA #45          ;SET PRINTER OFF CHAR IN
                          ;PRINT BUFFER
         JSR STRPRT
         JMP HOME         ;HOME CURSOR AND BEGIN AGAIN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FILE BLOCK MANAGEMENT

```
THIS ROUTINE MAINTAINS A FILE DIRECTORY ON
THE CURRENT TEXT WORD, AND
IS THE ONLY ROUTINE THAT ALTERS
THE PARAMETER CHRWD1.
FBMAN    PHA              ;SAVE A, X
         TXA
         PHA
         SEC              ;CALC LENGTH OF FILE,
                          ;E=LENGTH
         LDA TXTL         ;USING THE LEFTHAND
                          ;LIMIT
         SBC STXTL
         CMP MINF
         BCC RT           ;BRANCH IF E IS LESS
                          ;THAN THE MINIMUM
                          ;FILE LENGTH.
         LDX FBNDX
```

```
          INX             ;ADJUST INDEX TO 2'ND
                          ;ENTERY.
          STA FB,X        ;SAVE LENGTH IN
                          ;SECOND ENTERY.
          DEX
          LDA STXTL       ;SET LEFT LIMIT.
          STA FB,X        ;SET FIRST ENTRY
          INX
          INX
          STX FBNDX       ;SAVE INDEX TO FILE.
RT        CLC
          LDA TXTL        ;CALC FILE LENGHT RIGHT
                          ;SIDE.
          ADC CHRWD2
          STA TMP1        ;(TXTL+CHRWD).
          CLC
          LDA STXTL       ;(TXTL+CHRWRD2).
          ADC CHRWD1
          SEC
          SBC TMP1        ;N=(TXTL+CHRWD2)-(STXTL+CHRWD1)
          CMP MINF
          BCC END         ;BRANCH IF N LESS THAN MINF.
          LDX FBNDX
          INX             ;ADJUST FOR 2'ND ENTERY.
          STA FB,X        ;STORE IN THIS ENTERY.
          DEX
          LDA TMP1
          STA FB,X        ;TXTL+CHRWD2 INTO FIRST ENTERY
          INX
          INX
          STX FBNDX       ;REPLACE INDEX
END       PLA             ;RESTORE REGISTERS
          TAX
          PLA
          RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHANGE TEXT
REPLACE THE TEXT

THE LOWER CONTRACTION TO THE MOST RIGHT IN THE TEXT
WORD IS REPLACED BY ITS UNCONTRACTED FORM

```
REPLAC    LDY #00
BACK      LDA (LTABLE),Y ;GET BRAILLE CHARACTER
          STA (LTEXT),Y  ;STORE IT IN TEXT WORD
          INY
          CPY LCHRW2
          BNE BACK
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
THIS SUBROUTINE REPLACES THE CHARACTERS IN
THE TEXT WORD WITH BRAILLE CHARACTERS.
CHARACTERS EQUAL TO 'FF' ARE NON PRINTABLE
CHARACTERS.
CHGTXT   LDY CHRWD2      ;GET THE LENGTH
                         ;OF THE MATCH
         BIT CNTRL4      ;CHECK MODE WHETHER
                         ;WORDSIGN OR ABBREVIATION
         BVS OVR1        ;IF ABBREV MODE JUMP.
         INY
         INY
OVR1     STY TMP3        ;SAVE AS POINTER TO SYMBOLS.
         LDA #00
         STA TMP1
         STA TMP2
         STA TMP3
AGAIN    LDY TMP3
         LDA (TABLE),Y   ;GET SYMBOL
         ORA TMP2        ;TURN ON OR OFF ACCORDING
                         ;TO CNTRL WORD.
         INC TMP3        ;ADJUST TABLE INDEX.
         LDY TMP1        ;GET TABLE INDEX
         STA (TEXT),Y    ;STORE SYMBOL IN TEXT AREA
         INC TMP1        ;ADJUST TEXT INDEX
         AND #FF         ;SET STATUS FLAGS
         BPL AGAIN       ;LOOK FOR B7 ON IN SYMBOL
         LDA #FF         ;TURN ON NON PRINT FLAG
         STA TMP2
         LDA TMP1        ;GET NUMBER OF CHAR'S MOVED
         CMP CHRWD2      ;SEE IF MOVE IS COMPLETE
         BNE AGAIN
         LDY #00         ;TURN B7 ON
BACK     LDA #80         ;ALL TABLE CHARS TO BE PLACED
                         ;IN TEXT BUFFER
         ORA (TEXTL),Y
         STA (TEXTL),Y
         INY
         CPY CHRWD2
         BNE BACK
         RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

STORE IN PRINT BUFFER

```
THE CHARACTER IN A IS STORED IN THE PRINT BUFFER AND
THE POINTER TO THE BUFFER ADJUSTED
STRPRT   STY TMP0
         STA TMP1
         LDY #00
         CLC
         LDA PBFL        ;ADJUST INDIRECT POINTER
```

```
          ADC #01
          STA PBFL
          LDA TMP1
          STA (PBF),Y     ;STORE CHARACTER
          LDY TMP0
          RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

MAP

```
THE ASCII CHARACTER IN A IS MAPPED INTO A
BRAILLE SYMBOL
MAP       STY TMP1
          LDY =00
          ORA #20         ;ADJUST FOR UPPER
                          ;AND LOWER CASE
CMPR      CMP A760,Y      ;CHECK TABLE ENTRY
          BEQ OUT
          INY
          INY
          BNE CMPR
OUT       INY
          LDA A760,Y      ;GET BRAILLE SYMBOL
          LDY TMP1
          RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FILE TO PRINT BUFFER

```
THE PRINTABLE CHARACTERS IN THE TEXT BUFFER
ARETRANSFERRED TO THE PRINT BUFFER
FPRTBF    LDX #FF
BK        INX
          LDA 60,X
          CMP #FF
          BEQ REJECT
          AND #3F         ;MASK BITS 6&7
          JSR STRPRT      ;X MUST BE PRESERVED
REJECT    CPX #0F
          BEQ DONE
          JMP BK
DONE      RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WRITE

```
THE AREA POINTED TO BY X, AND Y IS CLEARED, A IS THE
CHARACTER TO BE WRITTEN
```

```
WRITE    STX VPNTL        ;X AND Y CONTAIN THE
                          ;POINTER TO THE AREA TO BE
                          ;CLEARED
         LDX #-3          ;FOUR PAGES WILL BE WRITTEN
         STY VPNTH        ;X IS LOW PART, Y IS HIGH
         LDY #00
BK       STA(VPNT),Y      ;STORE CHARACTER THRU POINTER
         INY
         BNE BK
         INC VPNTH
         INX
         BNE BK
         RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

COPY

```
A COPY OF THE VIDEO DISPLAY IS MADE IN RAM
COPY     JSR INVDO
AGAIN    CLC              ;ADJUST BY PLUS ONE ALL
                          ;POINTERS FOR TEXT
         LDA #01
         ADC CPNTL
         STA CPNTL
         STA VPNTL
         BCC OVR1
         LDA #00
         ADC CPNTH
         STA CPNTH
         INC VPNTH
         CMP #84          ;CHECK RANGE EXIT IF OUT OF RANGE
         BEQ OUT
CVR1     LDA (VPNT),Y     ;GET CHARACTER FROM SCREEN
         AND #7F          ;B7 OFF FOR COPY AREA
         TAX
         LDA (CPNT),Y
         ORA #80          ;B7 ON FOR VIDEO BOARD
         STA (VPNT),Y
         TXA
         STA (CPNT),Y
         JMP AGAIN
OUT      RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

RESTART

```
COPY THE BRAILLE FILE TO THE TEXT BUFFER AND REST FLAGS
RESTRT   JSR FPRTBF       ;COPY BRAILLE FILE TO PRINT
                          ;BUFFER
         JSR INTXT        ;INITIALIZE THE TEXT AREA
```

```
        LDA #F3
        AND FLAG
        STA FLAG
XIN     LDX #00         ;RESET TEXT INDEX FOR NUMBERS
        STX TMP1
        RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

RESET

```
COPY FILE TO PRINT BUFFER AND CLEAR TMP1
RESET   JSR FPRTBF
        JSR INTXT
        LDX #00
        STX TMPN1
        RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TERMINATION CHECK

```
A SPECIAL SYMBOL IS CHECKED FOR THE END OF THE
ENGLISH TEXT TO STOP FURTHER TRANSCRIPTION
SPEC2   ORA #80         ;TURN BIT 8 ON
        CMP #9D
        BNE OVRSP1
SP2BK1  JSR TRFB        ;EXIT IGNORE
        RTS
CVRSP1  CMP #92         ;IS IT A TERMINATING CHAR
        BNE SP2BK1
        LDA #01         ;THIS MARKS THE END OF THE
                        ;PAGE SIGNAL EXECUTIVE
        STA FLAG
        RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TERMINATION CHARACTER

```
A SPECIAL CHARACTER IS PLACED IN THE TEXT TO MARK
THE LAST POINT OF TRANSCRIPTION THAT IS TO OCCUR
TERM    LDA #12         ;STORE IT ON THE SCREEN
        JSR TXT+1
        JMP EXIT2
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

E. SEARCHING, SORTING AND CHECKING ROUTINES

CHECK

```
THE CONTRACTION FOUND IN THE SEARCH ROUTINE
IS CHECKED TO SEE IF IT IS AN
EXACT MATCH AND IF NOT IF AN OVERLAP
CCCURS WITH A PREFIX OR SUFFIX.
CHEQ     LDA LENGTH
         CMP CHRWD2
         BEQ OVR
         JSR SYLL
         CMP #FF
         BEQ ARND
         LDA #02          ;CODE IMPLIES NO OVERLAP
ARND     RTS              ;EXIT
OVR      LDA #01          ;EXACT MATCH
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PUNCTUATION SUCEEDING

```
SETS A FLAG TO INDICATE THAT PUNCTUATION
FOLLOWING THE WORD HAS BEEN FOUND
THEN TRANSCRIBES ONE WORD AND TRANSFERS
THE CHARACTERS TO THE PRINT BUFFER
PCNSUC   LDA #01          ;SET PUNCTUATION
                          ;SUCCEEDING FLAG
         ORA CNRL1
         STA CNTRL1
TRFB     JSR TRANS
FB       JSR FRTBF
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DECREMENT POINTER

```
MOVE THE POINTER TO THE CURRENT CHARACTER
IN THE COPY AREA BACK BY ONE
DECPNT   PHA              ;SAVE A
         SEC
         LDA CPNTL
         SBC #01
         STA CPNTL
         LDA CPNTH
         SBC #00
         STA CPTNH
         PLA
```

```
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PREVIOUS CHARACTER

THE PREVIOUS CHARACTER IS CHECKED AND A CODE
RETURNED TO INDICATE THE TYPE OF
CHARACTER FOUND.

```
PRECH     JSR DECPNT
          JSR DECPNT
CK1       JSR CHAR
CK        CMP #7B          ;IS IT A LETTER
          BCC OVR          ;BRANCH IF IN RANGE
          BCS NOTLTR       ;EXIT NOT A LTTR
OVR       CMP #61
          BCS LTTR
          CMP #5B          ;SEE IF A CAPITAL LETTER
          BCC OVR1
          BCS NOTLTR
CVR1      CMP #41
          BCS LTTR
NOTLTR    CMP PERIOD
          BEQ PER
          JSR CHAR         ;ADJ POINTER
          LDA #FF          ;SET TO SIGNIFY NOT A NUMBER
                           ;LETTER OR PERIOD
          RTS              ;EXIT
LTTR      JSR CHAR
          LDA #00
          RTS              ;SET A=00 TO INDICATE A
                           ;LETTER DETECTED
PER       JSR CHAR         ;ADJUST POINTER
          LDA #01          ;INDICATE A PERIOD
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHECK NEXT CHARACTER

THE NEXT CHARACTER IS FOUND AND CHECKED FOR TYPE

```
CHKNXT    JSR CK1
          JSR DECPNT
          JSR DECPNT
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++
```

PUNCTUATION PRECEEDING

```
A FLAG IS SET TO INDICATE THAT PUNCTUATION PRECEEDS
THE TEXT WORD
PCNPRE   LDA #10
         ORA CNTRL1
         STA CNTRL1
         JMP ONE          ;A JUMP IS MADE TO THE
                          ;PUNCTUATION ROUTINE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SORT

```
ALL PUNCTUATION IS PLACED IN BRAILLE ORDER BEFORE
BEING OUTPUT TO THE PRINT BUFFER
SORT     LDA CNTRL3
         AND #01          ;DOUBLE QUOTE?
         BEQ INNER
         LDA #26          ;STORE A DOUBLE QUOTE
         JSR STRPRT
INNER    BIT CNTRL3       ;INNER QUOTE?
         BPL BRKR
         LDA #20          ;STORE OPENING INNER QUOTE
         JSR STRPRT
         LDA #26
         JSR STRPRT
BRKR     BIT CNTRL3       ;OPENING ROUND BRACKET?
         BVC BRKSQ
         LDA #36          ;STORE OPENING ROUND BRACKET
BRKSQ    LDA CNTRL3       ;OPENING SQUARE BRACKET?
         AND #02
         BEQ LTTRS
         LDA #20          ;STORE OPENING SQUARE BRACKET
LTTRS    LDA CNTRL3       ;LETTER SIGN?
         AND #10
         BEQ APOS
         LDA #30          ;STORE LETTER SIGN
         JSR STRPRT
APOS     LDA CNTRL3       ;APOSTROPHE?
         AND #20
         BEQ CAPP
         LDA #04          ;STORE APOSTROPHE SIGN
         JSR STRPRT
CAPP     LDA CNTRL1       ;CAPITAL?
         AND #20
         BEQ JUST1
         LDA #20          ;STORE A CAPITAL LETTER SIGN
         JSR STRPRT
JUST1    LDA #20
         JSR STRPRT
CVR      LDA #80          ;ADJUST FLAG
         ORA CNTRL2
         STA CNTRL2       ;SET PUNCTUATION BUFFER
```

```
                        ;SORTED FLAG
        RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHARACTER

```
A CHARACTER IS OBTAINED FROM THE COPY AREA AND
RETURNED IN THE ACCUMULATOR
CHAR    STY TMP0
        LDY #00
        CLC
        LDA CPNTL       ;CALCULATE NEW POINTER
        ADC #01
        STA CPNTL
        LDA CPNTH
        ADC #00
        STA CPNTH
        CMP #84
        BEQ OVR         ;CHECK FOR OUT OF LIMITS
        LDA (CPNT),Y
CVR     LDY TMP0
        RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DASH SUCFEDING

```
CNE HALF OF A BRAILLE DASH IS STORED IN THE
PRINT BUFFER AND A FLAG IS SET
DSUC    JSR PNCSUC      ;SET FLAG
        LDA #24
        JSR STRPT       ;ONE-HALF DASH INTO
                        ;PRINT BUFFER
        RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DASH PRECEEDING

```
CNE-HALF OF A BRAILLE DASH IS STORED IN THE
PRINT BUFFER
DPRE    LDA #24
        JSR STRPRT
        JMP PNCPREC     ;SET A FLAG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHECK NUMBER

```
A CHECK IS MADE FOR A NUMBER IN A AND A CODE IS
RETURNED IN A TO INDICATE WHETHER OR NOT
A NUMBER HAS BEEN FOUND.
CKNUM     CMP #3A
          BCS OVR
          CMP #30
          BCS NM
OVR       LDA #FF          ;NOT A NUMBER
          RTS
NM        LDA #02          ;NUMBER FOUND
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SET POINTER

```
POINTERS FOR THE VALID AND INVALID ENTRY POINTS ARE SET
SETP      LDA #B3          ;INVALID=B3A1
          STA VLPH
          STA INVLDH
          LDA #D8          ;VALID=BED8
          STA VLDPH
          LDA #A1
          STA INVLDL
          RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
SETS ALL BITS IN CNTRL4 EXCEPT B3,4,6
CNTRL4    LDA #D8
          AND CNTRL4
          STA CNTRL4
          RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SYLLABLE OVERLAP

```
DETERMINES IF AN OVERLAP OCCURS BETWEEN THE PREFIX AND
OR SUFFIX AND THE MATCH FOUND.
SYLLAB    LDA #0F
          AND PCNTRL
          BNE P
CONT      LDA #0F
          AND SCNTRL
          BNE S
          BEQ ACCPT
P         STA TMP2         ;PCNTRL*0F
          LDA #0F
          CMP TMP2         ;IS MATCH ON RIGHT SIDE OF
                           ;PREFIX?
          BCS CONT
          CLC
```

```
          ADC CHRWD2
          CMP TMP2
          BCC CONT
          BEQ CONT
REJECT    LDA #FF          ;REJECT IT
          RTS              ;EXIT
S         STA TMP2         ;SCNTRL*0F
          LDA #0F
          AND TXTL
          STA TMP1         ;TXTL*0F
          LDA LENGTH
          SEC
          SBC TMP2
          CMP TMP1
          BCC ACCEPT
          BEQ ACCEPT
          JMP PSYLLA       ;JUMP TO PATCH
ACCEPT    LDA #01
          RTS              ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SEARCH FOR TWO CHARACTERS

THE CHARACTERS IN X AND A RESPECTIVELY ARE LOOKED FOR PRECEEDING THE CURRENT MATCH, A CODE IS RETURNED IN A TO REFLECT THE STATUS OF THE SEARCH.

```
DBLCHQ    JSR CHGPNT
          DEC TEXTL
          DEC TEXTL
          LDY TEXTL
          CPY #CF          ;CHECK LEFTHAND LIMIT
          BCC NOUT
          LDY #00
          CMP (TEXT),Y     ;IS FIRST LETTER OF SET EQUAL
                           ;TO THE FIRST IN THE SET
          BEQ EQ1
NOUT      INC TEXTL
          BNE OUT
EQ1       INC TEXTL        ;FIRST LETTER FOUND TO BE EQUAL
          TXA
          CMP (TEXT),Y
          BEQ EQ2
CUT       INC TEXTL        ;RESTORE TEXTL
          JSR CHGPNT
          LDA #FF          ;SET CODE TO REFLECT MATCH
                           ;AS NOT FOUND.
          RTS
EQ2       INC TEXTL
          JSR CHGPNT
          LDA #01
          RTS              ;SET CODE TO REFLECT A MATCH
                           ;FOUND
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 CHECK WHOLE WORD CONTRACTION

THE CURRENT TEXT WORD IS CHECKED TO SEE IF IT IS A
WHOLE WORD CONTRACTION,
CNTRL4 IS SET TO REFLECT THE RESULT
CNW      LDA CHRWD1
         CMP #5
         BCS NWW1          ;BRANCH IS >OR= TO 5
         CMP #4
         BCC THREE
         TAX               ;ITS FOUR LETTERS
                           ;CHECK FOR 'WITH'
 DEC     DEX
         BMI OUT           ;BRANCH IF LOOP COMPLETE
         LDA 60,X          ;GET TEXT CHARACTER
         CMP A61B,X        ;COMPARE TO TABLE
         BEQ DEC           ;CONTINUE IF MATCH
NWW1     JSR ADJC4         ;NOT A WWC RESET CNTRL4
OUT      RTS               ;EXIT
THREE    CMP #3            ;CHECK FOR 'AND, FOR'
         BCC TWO           ;BRANCH IF LESS THAN 2
         STA CNTR
MORE     LDX CNTR
         LDY TAB3,X        ;GET INDEX FOR TABLE
         LDX #3
AGAIN    DEY
         DEX
         BMI OUTM
         LDA 60,X          ;COMPARE TEXT TO TABLE
         CMP A600,Y
         BEQ AGAIN
         DEC CONTR         ;CHECK IF MORE WORDS LEFT
         BNE MORE
NWW2     JSR ADJC4
OUTM     RTS               ;EXIT
TWO      CMP #2            ;'OF'
         BCC ONE           ;BRANCH IF LESS THAN 2
         LDA #15           ;BRAILLE 0
         CMP 60
         BNE OUTN
         LDA #0B           ;BRAILLE 'F'
         CMP 61
         BNE NWW2
         RTS               ;EXIT
ONE      LDA #A
         CMP 60
         BNE NWW2
         RTS
```

TABLE 3

```
TAB3    AND     3E
        FOR     75
        THE     BB
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DIPHTHONG OR DIAERESE

THE PRESENCE OF A DIAERESE OR A DIPHTHONG IS LOOKED FOR AND A CODE IS RETURNED

```
DIPDIR  JSR CHGPNT      ;SET FOR UNCONTRACTED TEXT
        TXA
        PHA
        LDA TEXTL
        TAX
        CMP LFTPNT      ;CHECK FOR LEFT LIMIT
                        ;WHICH IS 60.
        BEQ ACCPT
        DEX
        STX TEXTL
        INX
CHEC    LDY #00
        LDA (TEXT),Y    ;GET A CHARACTER
        CMP #BRAILA     ;EQUIVALENT BRAILLE SYMBOL
                        ;FOR THE LETTER A.
        BEQ SYL
        CMP #BRAILO
        BNE NODIP
SYL     JSR SYLLAB
        CMP #FF
        BEQ ACCPT
REJECT  LDY #FF         ;REJECT NO OVERLAP
        BNE OUT
ACCPT   LDY #01         ;DIP OR DIR IS PART OF A
                        ;PREFIX OR SUFFIX.
        BNE OUT
NODIP   LDY #02         ;NO DIP OR DIR FOUND
OUT     STX TEXTL
        JSR CHGPNT
        PLA
        TAX
        TYA
        RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

DOT ONE AND FOUR CHECK

THE CONTRACTED TEXT WORD IS SEARCHED FOR AN UPPER DOT

```
DOT14    LDY CHRWD2     ;GET INDEX CLEAR MATCH
                        ;CHARACTERS
         DEY
BCK1     LDA (TEXT),Y   ;GET CHARACTER
         STA LC1,Y      ;SAVE IT
         LDA #00
         STA (TEXT),Y   ;CLEAR TEXT CHARACTER
         DEY            ;ADJUST INDEX
         BPL BCK1       ;CONTINUE UNTIL ALL DONE
END      INY            ;BEGIN LOOKING FOR UPPER DOTS
         CPY LENGTH     ;IS ENTIRE WORD DONE
         BEQ NDOT
         LDA 0060,Y
         CMP #FF
         BEQ END
         AND #09        ;SEE IF UPPER DOT EXISTS
         BEQ END
         LDA #01        ;FOUND AN UPPER DOT
STACK    PHA
         LDY CHRW2      ;RESTORE CONTRACTED AREA
         DEY
BK       LDA LC1,Y
         STA (TEXT),Y   ;REPLACE ORIGINAL CHARACTER
         DEY
         BPL BK
         PLA
         RTS            ;EXIT
NODOT    LDA #FF
         JMP STACK      ;EXIT AFTER RESTORATION
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTION CHECK

THE PRESENCE OF AN UPPER DOT IS CHECKED FOR, SHOULD
AN UPPER DOT NOT BE FOUND
THE TEXT WORD IS ADJUSTED SUCH THAT THE MOST RIGHTHAND
LOWER CONTRACTION ISLEFT UNCONTRACTED.

```
ICONCQ   TXA
         PHA
         LDA #00
         CMP LCHWD2     ;SEE IF PREVIOUS LOWER
                        ;CONTRACTION EXISTS
         BNE PREVLC
NEWONE   LDA #00        ;SET PARAM FOR LOWER
                        ;CONTRACTION
         STA LTEXTH
         LDA TEXTL
         STA LTEXTL
         LDA TABLEL
         STA LTBLEL
         LDA TABLEH
         STA LTBLEH
```

```
          LDA CHRWD2
          STA LCHRW2
          PLA
          TAX
          LDA #01          ;RETURN WITH ACCEPT CODE
          RTS
PREVLC    LDA TEXTL
          CMP LTEXTL
          BEQ NSWTCH
          BCS SWTCH
NSWTCH    JSR DOT14        ;CHECK FOR UPPER DOT
          CMP #01
          BEQ OVR
          JSR REPLAC
          JMP NEWONE
CVR       PLA
          TAX
          LDA #01          ;ACCEPT
          RTS
SWITCH    JSR DOT14
          CMP #01
          BEQ NEWONE
          PLA
          TAX
          LDA #FF          ;REJECT THIS IS MOST RT
                           ;CONTRACTION AND NO UPPER DOTS
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTION VOWEL CHECK

ONE OF THE FOLLOWING VOWELS A, E, O, I, U, Y IS
SEARCHED FOR AND A CODE RETURNED IN THE ACCUMULATOR
ACCORDING TO THE RESULT OF THE SEARCH.

```
LCVWL     LDY #00
MORVWL    LDA #0F          ;MASK OFF INDEX OF ROOT
          AND CHRWD2
          TAX              ;X CONTAINS STRT INDEX OF ROOT
          LDA VWLTAB,Y     ;GET VOWEL
CONTCQ    CMP 60,X         ;LOOK FOR VOWEL IN ROOT
          BEQ ACCEPT
          INX              ;HAVE NOT FOUND VOWEL YET
          CPX LENGTH
          BNE CONTCQ
          INY
          CPY #06
          BNE MORVWL
RTCT      LDA #FF          ;NC VOWEL FOUND
          RTS
ACCPT     LDA #01          ;FOUND A VOWEL
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SEARCH FOR BE PREFIX

```
THE VALUE IN THE ACCUMULATOR IS COMPARED TO A TABLE
A CODE RETURNED TO REFLECT THE RESULT
BESRCH   LDY#00
CK       CMP BETAB,Y
         BEQ ACCPT
         INY
         CPY #7
         BNE CK
RJCT     LDA #FF          ;NO MATCH FOUND
         RTS
ACCPT    LDA #01          ;MATCH FOUND ACCEPT CODE
         RTS
BETAB    A
         E
         I
         O
         U
         H
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SAVE TEXT

```
THE BRAILLE CODE OF THE TEXT WORD IS SAVED
SVTEXT   LDY #00
BK       LDA 0060,Y
         STA 00D0,Y
         INY              ;ADJUST INDEX
         CPY #10
         BNE BK
         LDA #00          ;INIT POINTER TO TEXT WORD
         STA TEXTH
         LDA #60
         STA TEXTL
         STA LFTPNT       ;SET LFTHND POINTER
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHANGE THE INDIRECT POINTER

```
CHANGE THE INDIRECT TEXT POINTER
CHGPNT   LDA #B0
         EOR TEXTL
         STA TEXTL
         LDA #B0
         EOR LFTPNT
         STA LFTPNT       ;SET MIN LEFT BOUNDARY
         RTS
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```

CHECK A AND RETURN CODE

```
THE CHARACTER IN A IS CHECKED AND A CODE RETURNED IN
A AS TO WHAT WAS FOUND
CK      CMP #FF
        BEQ FOUNDP      ;SEE IF OFF SCREEN AND
                        ;OUT OF RAM
        CMP #20         ;CHECK FOR A SPACE
        BNE OVR1
        LDA #00         ;NOT PCN NOT LETTER
        RTS
OVR1    CMP #41         ;CHECK FOR ALPHABET
        BCC OVR2
        LDA #01         ;SET CODE FOR ALPHABET
        RTS
OVR2    LDY #9
BACK    CMP PCNTAB,Y
        BEQ FOUNDP
        DEY
        BNE BACK
        LDA #00         ;DID NOT FIND PCN
        RTS
FOUNDP  LDA #FF         ;FOUND PCN
        RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PCNTAB
        =
        ,
        .
        ?
                :
        ;
        ,
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```

SEARCH TEXT

```
THE TEXT WORD IMMEDIATELY FOLLOWING THE CURRENT TEXT
WORD IS SEARCHED FOR ONE OF THE FOLLOWING WORDS:
AND, BUT, OR, WERE, WAS, IN, HIS, BY, ENOUGH.
A  CODE IS RETURNED IN A.
SRCHSET LDY INDX        ;INDX INDEX OF FIRST
                        ;CHARACTER OF NEXT WORD
        STY IND2        ;IND2 MUST BE SET TO TEXT
                        ;START AREA
NEWWRD  LDX #00
        STX INDX3       ;INDX3 IS THE INDEX TO
                        ;CHARACTER IN ROM WORD
```

```
            INC XINDX       ;INDEX TO TABLE OF LENGTHS
                            ;AND WORD
            LDY XINDX
            CPY #0A
            BEQ OUT         ;BRANCH IF SEARCH FINISHED
                            ;NO MATCH FOUND
            LDX WRDLNG,Y    ;GET SEARCH LENGTH INTO X
            LDY AINDX       ;AINDX IS INDEX TO TABLE
                            ;OF POINTERS
            LDA TBL,Y       ;SET INDIRECT POINTER TO TABLES
            STA PNTL
            INC AINDX
            LDY AINDX
            LDA TBL,Y
            STA PNTH
BK          INC AINDX       ;ADJUST INDEX
            LDY INDX2       ;GET CURRENT INDEX
            INC INDX2
            LDA (CPNT),Y    ;GET TEXT (ASCII)
            JSR MAP
            LDY INDX3
            INC INDX3
            CMP (PNT),Y
            BNE NEWWRD
            DEX             ;DECREASE BY ONE THE SEARCH
                            ;LENGTH
            BNE BK
            LDA #01         ;FOUND A MATCH
            RTS
OUT         LDA #FF         ;DID NOT FIND A MATCH
            RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
TBL         AND
            BUT
            OR
            WERE
            WAS
            IN
            HIS
            BY
            ENOUGH
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
WRDLNG      03
            03
            02
            04
            03
            02
            03
            03
            02
            06
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

CHECKA

```
THE ASCII CHARACTER IN A IS CHECKED AND A CODE
RETURNED TO INDICATE ITS TYPE
CHECK    TAY              ;SAVE CHARACTER
         CMP #7B
         BCS SPEC
         CMP #61
         BCS LTTRLC
         CMP #5B
         BCS SPEC2
         CMP #41
         BCS LTTRUP
         CMP #3A
         BCS SPEC3
         CMP #30
         BCS NUM
         CMP #2E
         BCS APPOS
PERIOD   LDA #07
         RTS
SPEC1    LDA #01
         RTS
SPEC2    LDA #02
         RTS
SPEC3    LDA #03
         RTS
NUM      LDA #04
         RTS
LTTRLC   LDA #05
         RTS
LTTRUP   LDA #06
         RTS
APPOS    CMP #'           ;APPOST
         BNE OUT
         LDA #08          ;IT WAS APPOST
         RTS
OUT      LDA #09
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LAST CHARACTER CHECK

```
CHECK THE LAST CHARACTER AND SEE WHAT IT IS, RESTORE
EVERYTHING ON EXIT
LAST     JSR DECPNT       ;GET PAST CHARACTER
         JSR DECPNT
         JSR CHAR
         JSR CHECK        ;SEE WHAT IT IS
         PHA              ;SAVE CODE
         JSR CHAR
```

```
          PLA             ;GET CODE
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PAST CHARACTER

```
THE PRINT BUFFER POINTER IS LEFT POINTING TO THE
PREVIOUS CHARACTER IN
THE
BUFFER, THE CHARACTER IS RETURNED IN A.
PAST      LDY #00
          LDA (PBF),Y     ;GET LAST CHARACTER STORED
          PHA
          SEC
          LDA PBFL        ;ADJUST POINTER
          SBC #01
          STA PBFL
          LDA PBFH
          SBC #00
          STA PBFH
          PLA
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PREFIX

```
DETERMINE IF A PREFIX FROM THE TABLE EXISTS AT THE
START OF THE CURRENT TEXT WORD, THE ASSUMED ROOT WORD
IS THREE LETTERS LONG AND MUST HAVE A VOWEL
SUFJMP    JMP SUFFIX      ;OUT OF RANGE JUMP
PREFIX    PHA
          TYA
          PHA
          TXA
          PHA
          LDA #40         ;INITIALIZE CONTROL WORDS
          STA PCTRL       ;SET B6=1 FOR PREFIX VOWEL
                          ;SEARCH
          STA SCTRL
          LDA LENGTH      ;GET WORD LENGTH
          STA END         ;SET END OF SEARCH
          SEC
          SBC #4          ;ROOT=PREFIX+3
          BMI OUT
          CLC
          ADC #01         ;ADJUST
          LDY #06
          STY OFF         ;INITIALIZE START POINT FOR
                          ;ROOT CHECK
          CMP #06
```

```
          BCS POVR1          ;IN RANGE STORE CALCULATED
                              PREFIX LENGTH IF A LESS THAN 6
          STA OFF
POVR1     JSR VOWEL
PSTRT     LDY OFF            ;CHECK IF A VOWEL WAS FOUND
                             ;IN THE ROOT
          TYA
          CMP #02            ;SEE IF PREFIX >,=2
          BCC SUFFIX
          LDA #A8            ;SET PNTR
          STA TABLE H
          LDA PREADD,Y       ;GET PNTRL
          STA TABLEH
          CMP #39
          BNE POVR2
          INC TABLEH
POVR2     LDX PRETAB,Y       ;GET NUMBER ENTERIES THIS
                             ;LENGTH
INIT      LDY #00
PAGAIN    LDA (TABLE),Y      ;GET DATA
          CMP 0060,Y
          BNE POVR3          ;BRANCH IF MISMATCH
          INY
          CPY OFF
          BNE AGAIN          ;FALL THRU WHEN WHOLE MATCH
          LDA OFF            ;SET PREFIX LENGTH INTO A
          ORA #80
          ORA PCNTRL
          STA PCNTRL
          JMP SUFFIX         ;CHECK FOR SUFFIX
POVR3     DEX
          BNE TABADJ         ;ALL ENTERIES SEARCHED?
          DEC OFF            ;YES DECREASE SEARCH LENGTH
          JMP PSTRT
TABADJ    LDA TABLEL         ;MORE ENTERIES THIS TABLE
          CLC
          ADC OFF            ;ADJ LOW BYTE
          STA TABLEL
          BCC INIT ;CO SET Y=00
          INC TABLEH         ;ADJUST HIGH BYTE BOUNDARY
                             ;CROSSED
          JMP INIT
OUT       PLA
          TAX
          PLA
          TAY
          PLA
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

VOWEL

```
THE ASSUMED ROOT IS SEARCHED FOR A VOWEL
VOWEL    PHA             ;SAVE REGISTERS
         TYA
         TXA
         PHA
BK5      LDX #6          ;GET TOTAL NUMBER OF VOWELS
BK3      LDY OFF         ;GET START POINT OF SEARCH
BK1      LDA VWL,X       ;GET A VOWEL
BK2      CMP 0060,Y      ;IS IT A VOWEL
         BNE OVR
BK4      LDA #40         ;TURN OF PREFIX MODE FLAG
         EOR PCNTRL
         STA PNCTRL
         PLA             ;RESTORE REGISTERS
         TAX
         PLA
         TAY
         PLA
         RTS             ;EXIT
OVR      INY
         CPY END         ;HAS END OF ROOT BEEN REACHED?
         BNE BK2
         DEX
         BNE BK3         ;CONTINUE USING A NEW VOWEL
         BIT PCNTRL
         BVC SUFCON      ;CHECK FLAG FOR SUFFIX OR
                         ;PREFIX MODE
         LDY OFF         ;OFF CONTAINS LENGTH OF PREFIX
         STY END
         DEY             ;OFF =START POINT OF SEARCH
         STY OFF
         BNE BK5
         JMP BK4         ;IF OFF=0 NO PREFIX ALLOWED
SUFCON   LDY END         ;END REALLY POINTS TO NEXT
                         ;CHARACTER THAT IS UNCHECKED
         STY OFF         ;CHOSE NEW STARTING POINT
         INY
         STY END         ;GET NEW END POINT, SEARCH
                         ;ONE AT A TIME
         CPY LENGTH
         BNE BK5
         JMP BK4
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
VWL      A
         E
         I
         O
         U
         Y
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SUFFIX

```
THE TEXT WORD IS CHECKED FOR A POSSIBLE PRESENCE OF
A SUFFIX
SUFFIX    LDA #0F
          AND PCNTRL
          STA OFF          ;GET PREFIX IF ANY
          LDY #06          ;ASSUME MAX SUFFIX
S1        LDA LENGTH
          STY TMP1
          SEC
          SBC TMP1
          STA END
          SBC OFF
          BMI SOVR1
          CMP #3           ;BRANCH IF ROOT >,=3
          BCS S2
SOVR1     DEY              ;ROOT IS NOT MIN LENGTH
          CPY #2           ;BRANCH IF Y>,=2
          BCS S1
JOUT1     JMP OUT
S2        JSR VOWEL
SSTRT     LDA LENGTH
          SEC              ;CALCULATE SUFFIX LENGTH
          SBC END
          STA TMP1
          CPA #2
          BCC JOUT
          CPA #7
          BCC SOVR1
          LDA #6
          STA TMP1
SOVR1     LDY TMP1         ;SUFFIX LENGTH INTO INDEX
          LDA #A9          ;SET TABLE POINTERS FOR SUFFIX
          STA TABLEH
          LDA SUFADD,Y
          STA TABLEL
          CMP #08
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PUNCTUATION ROUNTINE

```
THE AREA SURROUNDING THE TEXT WORD IS SEARCHED FOR
PUNCTUATION, AND FLAGS
SET TO REFLECT WHICH PUNCTUATION IS PRESENT.
PNC1J     JMP PCN1
PNC2J     JMP PNC2
SQBJ      JMP SQB
PCN       LDY #00          ;INITIALIZE THE CONTROL WORDS
          STY CNTRL1
          STY CNTRL2
```

```
                STY CNTRL3
                STY CNTRL5      ;CHR/WD1
                STY WDNX        ;INDEX FOR CURRENT WORD FILE
                LDA CHRWD1      ;GET LENGTH
                CMP #10         ;DO LENGTH CHECK
                BCS OUTR        ;BRANCH IF GREATER THAN 10
                LDA CPNTH
                CMP #FF
                BEQ OVR
CH              JSR CHAR        ;GET CHAR
OUT1            JMP ONE
OVR             LDA #01         ;NOTE END OF PAGE REACHED
                STA FLAG
                RTS
OUTR            JSR TRFB        ;MAX LENGTH WORD GO TRANSCRIBE
                RTS
ONE             STA TMP5        ;SAVE INPUT
                CMP #7B         ;BEGIN ASCII DECODING
                BCS SPECJ
                CMP #61
                BCS E           ;BRANCH IF LETTER
                CMP #5B
                BCS SQBJ
                CMP 41
                BCS CAP
                CMP #3A
                BCS PCN1J
                CMP #30
                BCS NUMJ
                CMP #1F
                BCS #PCN2J
SPECJ           JMP SPEC2
NUMJ            JMP NUM
CAP             LDA #40         ;BEGIN CAPITAL ROUTINE
                ORA CNTRL1
                STA CNTRL1      ;SET CAPITAL FLAG
E               INC CNTRL5      ;ADJUST WORD LENGTH
                BIT CNTRL1      ;2'ND LETTER OR GREATER?
                BMI LTTR2       ;BRANCH IF NOT FIRST LETTER
                LDA #80         ;SET FIRST LETTER FLAG
                ORA CNTRL1
                STA CNTRL1
                BIT CNTRL1
                BVS CAP1        ;BRANCH IF FIRST LETTER A
                                ;CAPITAL
                BIT CNTRL2      ;CHECK PCNBUFFER FLAG
                BMI FIVE
                JSR SORT        ;SORT PUNCTUATION BUFFER
FIVE            LDA TMP5        ;GET I/P
                JSR STRWD       ;PUT IN TEXT FILE
                LDA #BF         ;RESET CURRENT CAPITAL FLAG
                AND CNTRL1
                STA CNTRL1
                LDA #02
```

```
         ORA CNTRL1
         STA CNTRL1      ;SET TRANSCRIBE FLAG
         JMP ONE
CAP1     LDA #18
         ORA CNTRL1      ;PNC PREC, AND CAP DETECTED
         STA CNTRL1
         JMP FIVE
LTTR     BIT CNTRL1      ;NOT 1'ST LETTER
         BVS ALLCAP      ;IF FLAG SET THEN BRANCH
         BIT CNTRL2      ;SEE IF PCN NEEDS SORTING
         BMI OVRP1
         JSR SORT
OVRP1    JMP FIVE        ;EXIT
ALLCAP   LDA #38         ;SET ALL CAPITALS FLAG
         ORA CNTRL1      ;SET PNC PREC FLAG
         STA CNTRL1
         BIT CNTRL2      ;CHECK PCN BUFFER SORT FLAG
         BMI OVRP2
         JSR SORT        ;BRANCH IF ALREADY SORTED
OVRP2    JMP FIVE
SQB      CMP #5B         ;CHECK LEADING SQUARE BRACKET
         BNE ESQB
         LDA #02         ;SET FLAG
         ORA CNTRL3
         STA CNTRL3
         JMP PNCPREC     ;SET PNC PREC FLAG
ESQB     MP #5D          ;CHECK CLOSING SQUARE BRACKET
         BEQ SQBC
         JMP SPEC
SQBC     LDA #02         ;CHECK TRANS FLAG
         AND CNTRLQ      ;IS IT CLOSING BRACKET
         BEQ NOTRAN
         JSR SORT
         JSR PNCSUC      ;SET FLAG
         LDA #36         ;STORE CLOSING BRACKET
         JSR STRPRT
         LDA #04
         JSR STRPRT
         RTS             ;EXIT ALL CONTROL WORDS MUST
                         ;BE REINITIALIZED
NOTRAN   LDA #36         ;TRANS DONE
         JSR STRPRT
         LDA #04
         JSR STRPRT
         JMP ONE         ;GET NEXT CHARACTER
PCN2     CMP #20
         BNE PRIOD
         LDA #02         ;ITS A SPACE
         AND CNTRL1      ;CHECK TRANS FLAG
         BEQ SPB
         JSR TRFB        ;TRANSCRIBE AND STORE
         LDA #40
         JSR STRPRT      ;PLACE IN PRINT BUFFER
         LDA PBFL        ;SAVE A POINTER TO THE LAST
```

```
                        ;SPACE STORED
        STA SPNTL
        LDA PBFH
        STA SPNTH
        LDA #01
        AND FLAG        ;SEE IF SPACE IS TO BE STORED
        BEQ RTS
        EOR FLAG        ;RESET FLAG
        STA FLAG
        LDA #48
        LDY #00
        STA (SPNT),Y
RTS     RTS             ;EXIT
SPB     LDA #40         ;TRANS FLAG NOT SET
        JSR STRPRT      ;SPACE INTO PRT BUFFER
        LDA PBFL        ;SET UP SPACE POINTER
        STA SPNTL
        LDA PBFH
        STA SPNTH
        JMP ONE         ;GET A CHARACTER
PERIOD  CMP #2E         ;IS IT A PERIOD
        BNE COMMA
        LDA #02
        AND CNTRL1
        BEQ DOTE        ;IF TRANS FLAG SET BRANCH
        JSR PNCSUC
DOTE    JSR CHKNXT      ;GET NEXT TEXT SYMBOL
        CMP #01
        BNE PER
        LDA #04         ;STORE AN ELLIPSE
        JSR STRPRT
        JSR STRPRT
        JSR STRPRT
BKTHR   JSR CK1         ;CHECK NEXT CHARACTER
        JSR DECPNT      ;ADJUST PNTR
        CMP #01
        BEQ BKTHR       ;LOCP UNTIL ALL PERIODS
                        ;ACCOUNTED FOR
        PTS             ;EXIT
PER     LDA #32         ;STORE A PERIOD
        JSR STRPRT
        RTS             ;EXIT
COMMA   CMP #2C
        BNE EXCL
        LDA #02         ;CHECK TRANS FLAG
        ORA CNTRL1
        BEQ CB          ;BRANCH IF RESET
        JSR PNCSUC      ;SET FLAGS AND TRANSCRIBE
        LDA #02
        JSR STRPRT      ;COMMA INTO PRINT BUFFER
        RTS             ;EXIT
CB      LDA #02         ;COMMA INTO PRINT BUFFER
        JSR STRPRT
        JMP ONE         ;GET A CHARACTER
```

```
EXCL    CMP #21         ;IS IT EXCLAMATION POINT
        BNE NS          ;IF NOT BRANCH
        LDA #02         ;CHECK TRANS FLAG
        AND CNTRL1
        BEQ EXC
        JSR PNCSUC      ;SET FLAG AND TRANSCRIBE
        RTS             ;RESET CNTRL WORDS
EXC     LDA #16         ;STORE EXCLAMATION POINT INTO
                         PRINT BUFFER
        JSR STRPRT
        JMP ONE         ;GET NEXT CHARACTER
NS      CMP #23         ;IS IT A '#'
        BNE DOL
        JMP SPEC
DOL     CMP #24
        BNE QUOTE
        JMP DOL$
QUOTE   CMP #22
        BNE DASH
        LDA #02         ;CHECK TRANS FLAG
        AND CNTRL1
        BEQ OVRP3
        JSR PNCSUC      ;MUST BE A CLOSING QUOTE
        LDA #34
        JSR STRPRT
CVRP3   LDA #01
        ORA CNRTL3      ;SET QUOTE FLAG
        STA CNTRL3
        JMP PNCPRE      ;SET PREC FLAG
DASH    CMP #2D         ;SEE IF DASH
        BNE APOST
        LDA #08
        ORA CNTRL2
        STA CNTRL2
        LDA #02
        AND CNTRL2
        LDA #02
        AND CNTRL1
        BEQ DASHB
        JSR DSUC        ;POINT TO DASH AGAIN
        JSR DECPNT
        RTS
        JSR PREVCH      ;FIND OUT WHAT IT IS
        CMP #00
        BNE OVRP4       ;BRANCH IF NO LETTER DETECTED
        JMP DPRE        ;PREV CHAR WAS A DASH
OVRP4   LDA #24
        JSR STRPRT
        JMP DPRE        ;STORE LAST HALF OF DASH
APOST   CMP #27         ;SEE IF AN APOSTROPHE
        BNE OPENQ
        LDA #02         ;CHECK TRANS FLAG
        AND CNTRL1
        BEQ CLOSQ
```

```
        JSR CHKNXT      ;SEE WHAT NEXT CHAR IS
        CMP #00
        BEQ APOSTR      ;BRANCH IF ITS A LETTER
        JSR PNCSUC      ;ITS A CLOSING QUOTE
        LDA #34         ;STORE IT
        JSR STRPRT
        LDA #04
        RTS
APOSTR  LDA #20         ;SET FLAG
        ORA CNTRL3
        STA CNTRL3
        JSR RESET       ;CLEAR TRANS FLAG
        LDA #04         ;APOSTROPHE INTO PRINT BUFFER
        JSR STRPRT
        LDA #20         ;SET APOST FLAG
        ORA CNTRL3
        STA CNTRL3
        JMP PNCPRE      ;SET PRECEEDING FLAG AND
                        ;GET ANOTHER CHARACTER
OPENQ   CMP #2C         ;IS IT AN OPENING QUOTE
        BNE RNDBRC
        LDA #80         ;SET FLAG
        ORA CNTRL3
        STA CNTRL3
        JMP PNCPREC     ;SET FLAGS AND GET NEXT
                        CHARACTER
RESET   JSR PNCSUC      ;TRANS TO PRINT BUFFER
        LDA #FD         ;CLEAR TRANS FLAG
        AND CNTRL1
        STA CNTRL1
        JSR INTXT       ;CLEAR TEXT BUFFER
        RTS             ;EXIT
RNDBRC  CMP #29
        BNE RNDBRO
        LDA #02         ;ITS A CLOSING BRACKET
        AND CNTRL1
        BEQ RNDB        ;BRANCH IF TRANS FLAG RESET
        JSR PNCSUC
        LDA #36
        JSR STRPRT      ;PUT INTO PRINT BUFFER
RNDB    LDA #36
        JSR STRPRT
        RTS             ;EXIT
RNDBRO  CMP #28         ;SEE IF AN OPENING BRACKET
        BNE PERCNT
        LDA #40         ;SET FLAG
        ORA CNTRL3
        STA CNTRL3
        JMP ONE         ;GET A CHARACTER
PERCNT  CMP #25         ;SEE IF A PER CENT SIGN
        BNE SLASH
        LDA #02         ;CHECK TRANS FLAG
        AND CNTRL1
        BEQ PERCT       ;BRANCH IF TRANS FLAG RESET
```

```
                LDA #12         ;PER CENT SIGN INTO BUFFER
                JSR STRPRT
                LDA #0F
                JSR STRPRT
                LDA #3C         ;NUMBER SIGN INTO PRINT BUFFER
                JSR STRPRT
                JMP ONE         ;GET ANOTHER CHARACTER
SLASH           CMP #2F         ;SEE IF SLASH
                BNE ASTER
                LDA #02         ;CHECK TRANS FLAG
                AND CNTRL1
                BEQ ND          ;BRANCH IF RESET
                JSR TRFB
                LDA #0C         ;PUT INTO PRINT BUFFER
                JSR STRPRT
                RTS             ;RESET ALL CNTRL WORDS
ND              LDA #0C
                JSR STRPRT
                JMP ONE         ;GET CHARACTER
ASTER           CMP #2A         ;SEE IF ASTERISK
                BNE SPEC
                LDA #02
                AND CNTRL1
                BEQ ASTR
                JSR TRFB
ASTR            LDA #14         ;INTO PRINT BUFFER
                JSR STRPRT
                LDA #14
                JSR STRPRT
                RTS             ;EXIT
PCN1            CMP #3A         ;SEE IF A COLON
                BNE SCOL
                LDA #02         ;CHECK TRANS FLAG
                AND CNTRL1
                BEQ COLON
                JSR PNCSUC
COLON           LDA #12
                JSR STRPRT
                RTS             ;RESET CONTROL WORDS
SCOL            CMP #3B         ;SEE IF A SEMICOLON
                BNE QUEST
                LDA #02         ;CHECK TRANS FLAG
                AND CNTRL1
                BEQ SEMI
                JSR PNCSUC
SEMI            LDA #06
                JSR STRPRT
                RTS
QUEST           CMP #3F         ;SEE IF A QUESTION MARK
                BNE AT
                LDA #02
                AND CNTRL1
                BEQ QUESTN
                LDA #20         ;SET FLAG
```

```
         ORA CNTRL2
         STA CNTRL2
         JSR PNCSUC
QUESTN   LDA #26
         JSR STRPRT
         RTS             ;EXIT
AT       CMP #40         ;SEE IF
         BNE SPEC
         LDA #01
         JSR STRPRT
         LDA #1E
         JSR STRPRT
         RTS             ;EXIT
NUM      STA TMP0
         JSR RESET       ;O/P ANY INFORMATION IN TEXT
                         ;BUFFER TO PRINT BUFFER
         LDA TMP0
         CMP #24         ;IS IT A $
         BNE OVRN1
         LDA #32
         JSR STRPRT
         LDA #3C         ;STORE A NUMBER SIGN
         JSR STRPRT
         JMP FOUR
OVRN1    LDA #3C         ;STORE A NUMBER SIGN
         JSR STRPRT
         JSR LAST        ;LOOK FOR AN APOSTROPHE
         CMP #08
         BNE PERIOD
         JSR PAST
         LDA #04
         BNE FORW
PERIOD   CMP #07
         BNE TWO
         LDA #28         ;PERIOD FOUND SAVE PERIOD
FORW     PHA
         JSR PAST
         JSR PAST
         LDA #3C         ;NUMBER SIGN INTO PRINT BUFFER
         JSR STRPRT
         PLA
         JSR STRPRT
TWO      LDA TMP0        ;GET I/P
         CLC
         ADC #10
         CMP #40
         BNE OVRN2       ;CORRECT 0 TO J
OVRN2    JSR MAP
         LDX TMP1        ;GET TEXT INDEX
         INC TMP1
         STA TEXT,X
         CPX #0F         ;IF TEXT BUFFER FULL EXIT
         BNE FOUR
         JSR FPRTBF
```

```
        LDA #F3         ;A NUMBER SIGN HAS BEEN STORED
                        ;ALREADY, RESET FLAGS
        AND FLAG
        STA FLAG
        RTS
FOUR    JSR CHAR        ;GET A CHARACTER
        STA TMP0
        CMP #2E
        BNE OVRN3
        JSR FPRTBF
        JSR INTXT
        JSR XIN
        LDA #28
        JSR STRPRT      ;DECIMAL POINT INTO
                        ;PRINT BUFFER
        JMP FOUR
OVRN3   JSR CHECK       ;IS IT A NUMBER
        CMP #04         ;NUMBER CODE=04
        BEQ TWO
COMMA   LDA TMP0        ;GET I/P
        CMP #2C
        BNE SLSH
        JSR RESTRT
        LDA #02
        JSR STRPRT      ;COMMA INTO PRINT BUFFER
        JMP FOUR
SLSH    CMP #2F
        BNE CLN
        JSR RESET
        LDA #0C
        JSR STRPRT
        LDA #08         ;INDICATE THAT A FRACTION
                        ;SIGN RECEIVED
        ORA FLAG
        STA FLAG
        JMP FOUR
CLN     CMP #3A
        BNE TIL
        JSR RESET
        LDA #12
        JSR STRPRT
        JMP FOUR
TIL     CMP #7E
        BNE DSH
        JSR RESET       ;FRACTION DETECTED
        LDA #24
        JSR STRPRT
        JMP FOUR
DSH     CMP #2D         ;IS IT A DASH
        BNE PRCENT
        LDA #08         ;SEE IF FRACTION FLAG SET
        AND FLAG
        BNE MIXED       ;IF SET BRANCH ITS A
                        ;MIXED NUMBER
```

```
          JSR CHAR        ;GET NEXT NUMBER
          CMP #2D
          BEQ HYP
          JSR DECPNT
          JSR RESET
          LDA #24         ;STORE HYPHEN
          JSR STRPRT
          JMP FOUR
MIXED     JSR RESTRT
          LDA #24
          JSR STRPRT
          LDA #24
          JSR STRPRT
          RTS
HYP       JSR RESTRT
          LDA #24
          JSR STRPRT
          LDA #24
          JSR STRPRT
          RTS
PRCENT    JMP PRCNT       ;OUT OF RANGE BRANCH
PRCNT     CMP #25
          BEQ PCENT
OUT       JSR RESTART
          JSR DECPNT
          RTS
PCENT     LDA #FF         ;PUT MARKER ON STACK
BKN1      PHA
          JSR PAST        ;GET PREVIOUS CHAR FROM
                          ;PRINT BUFFER
          CMP #3C
          BEQ FNDIT
          JMP BKN1
FNDIT     JSR PAST
          CMP #32         ;IF PERIOD FOUND DO NOT RESTORE
          BEQ OVRN4
          JSR STRPRT
CVRN4     LDA #0F
          JSR STRPRT
        LDA #3C
BKN2      JSR STRPRT
          PLA
          CMP #FF
          BNE BKN2
          JSR RESTRT      ;TEXT TO PRINT BUFFER
          RTS             ;EXIT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

NUMBER ROUTINE

```
ALL NUMERALS AND SYMBOLS RELATED TO NUMERALS ARE
SORTED AND PLACED INTO THE PRINT BUFFER
NUM     STA TMPN0       ;ENTERY HERE ONLY OCCURS
                        ;WHEN A NUMBER IS DETECTED
        JSR RESET       ;O/P ANY INFORMATION IN TEXT
                        ;BUFFER TO PRTBFFR
        LDA TMPN0       ;GET I/P
        CMP #24         ;IS IT A DOLLAR SIGN?
        BNE OVRNM1
        LDA #32         ;DOLLAR SIGN MUST PRECEED
        JSR STRPRT      ;STORE IN PRTBFFR
        LDA #3C
        JSR STRPRT
        JMP NFOUR
CVRNM1  LDA #3C         ;NUMBER SIGN INTO PRINT BUFFER
        JSR STRPRT
        JSRLAST         ;GET LAST CHAR IN PRTBFFR AND
                        ;SEE WHAT IT IS
        CMP #08         ;CHECK FOR APPOST
        BNE NPERD
        LDA #04         ;PUT APPOST INTO BFFR
        JSR PAST        ;APPOST IS DETECTED AS
                        ;CLOSING QUOTE
        JMP OVRNM2
NPERD   CMP #07         ;IS IT A PERIOD
        BNE NMTWO
CVRNM2  LDA #28         ;IT WAS A PERIOD PUT ON STACK
        PHA
        JSR PAST
        JSR PAST
        LDA #3C
        JSR STRPRT
        PLA
        JSR STRPRT
NMTWO   LDA TMPN0       ;GET I/P
        CLC
        ADC #10
        CMP #40         ;CORRECT ZERO
        BNE OVRNM3
        LDA #4A
CVRNM3  JSR MAP
        LDX TMPN1       ;GET TEXT
        INC TMPN1       ;GET INDEX
        STA TEXTL,X
        CPX #0F         ;IF TEXT BUFFER FULL EXIT
        BNE NFOUR
        JSR FPRTBF
        LDA #F3         ;A NUMBER SIGN HAS BEEN
                        ;ALREADY STORED
        AND FLAG        ;RESET THE FLAGS
        STA FLAG
        RTS
NFOUR   JSR CHAR        ;GET A CHAR
```

```
            STA TMPNO
            CMP #2E          ;IS IT A DECIMAL PT
            BNE OVRNM4
            JSR FPRTBF
            JSR INTXT
            JSR XIN          ;RESET INDEX FOR TEXT BUFFER
            LDA #28          ;STORE A DECIMAL PT
            JSR STRPRT
            JMP NFOUR
CVRNM4      JSR CHECK        ;IS IT A MUMBER?
            CMP #04
            BEQ NMTWO
NCOMMA      LDA TMPNO        ;GET I/P
            CMP #2C          ;IS IT A COMMA?
            BNE NSLSH
            JSR RESTRT
            LDA #02
            JSR STRPRT
            JMP NFOUR
NSLSH       CMP #2F          ;IS IT A SLASH?
            BNE NCOLON
            JSR RESET
            LDA #0C          ;PUT SLASH INTO BFFR
            JSR STRPRT
            LDA #08          ;INDICATE A FRACTION RECEIVED
            CRA FLAG
            STA FLAG
            JMP NFOUR
NCOLON      CMP #3A          ;IS IT A COLON?
            BNE APPRX        ;USED FOR TIME
            JSR RESET        ;DO NOT RESET FLAGS
            LDA #12
            JSR STRPRT
            JMP NFOUR
APPRX       CMP #7E          ;IS IT APPRX SIGN?
            BNE NDSH
            JSR RESET
            LDA #24
            JSR STRPRT
            JMP NFOUR
NDSH        CMP #2D          ;IS IT A DASH
            BEQ OVRNM5
            JMP NPRCNT
CVRNM5      LDA #08          ;SEE IF FRACTIONAL SIGN SET
            AND FLAG
            BNE MIXED
            JSR CHAR         ;GET NEXT CHAR
            CMP #2D
            BEQ MIXED        ;STORE HYPHEN THEN EXIT
            JSR DECPNT       ;IT WAS NOT A DOUBLE DASH
            JSR RESET
            LDA #24
            JSR STRPRT
            JMP NFOUR
```

```
MIXED    JSR RESTRT      ;MIXED NOS MUST HAVE
                         ;DOUBLE 24''S STORED
         LDA #24
         JSR STRPRT
         JSR STRPRT
         RTS
NPRCNT   CMP #25         ;IS IT A PERCENT SIGN
         BEQ NPRCT
NOUT     JSR RESTRT
         JSR DECPNT
         RTS
NPRCT    LDA #FF         ;PUT MARKER ON STACK
NUMBK1   PHA
         JSR PAST        ;GET PREV CHAR FROM PRTBFFR
         JSR PAST
         CMP #3C         ;LOOK FOR NUMBER SIGN
         BEQ FNDIT
         JMP NUMBK1
FNDIT    JSR PAST
         CMP #32         ;IF A PERIOD IS FOUND DO NOT
                         ;RESTORE
         BEQ OVRNM6
         JSR STPRT
CVRNM6   LDA #12         ;STORE PERCENT SIGN
         JSR STRPRT
         LDA #0F
         JSR STRPRT
         LDA #3C
NUMBK2   JSR STRPRT
         PLA
         CMP #FF
         BNE NUMBK2
         JSR RESTRT
         RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SEARCH ROUTINE

```
THE CURRENT TEXT WORD IS SEARCHED FOR ANY OF THE
ABBREVIATION AND OR CONTRACTIONS THAT MAY APPLY,
THE MATCHS THAT ARE FOUND ARE CHECKED AND ACCEPTED
OR REJECTED ACCORDING TO THE RULES OF BRAILLE
SEARCH   LDA #00
         STA LCHR/WD2    ;INITIALIZE
         JSR SETP        ;SET RULE POINTERS
         JSR SVTEXT      ;SAVE THE TEXT WORD
         JSR CNW         ;ADJUST CNTRL4 FOR WORDSIGNS
         BIT CNTRL2      ;SORT PCN?
         BMI ARNDS1
         JSR SORT        ;YES
ARNDS1   LDA CHRWD1      ;CHECK LENGTH OF TEXT
         STA LENGTH
```

```
            CMP #02
            BCS GRTEQ       ;BRANCH IF >,=2
            LDA 60          ;NO MUST BE =01
            CMP #01
            BCS OVRS1
            JMP WS          ;JMP TO CHECK FOR AN A
OVRS1       JSR FBPRT       ;PUT INTO PRINT BUFFER
            JSR INTXT       ;INITIAL TEXT BUFFER
            JSR ADJC4       ;RESET CNTRL4
            RTS             ;EXIT
GRTEQ       BNE GRT         ;BRANCH IF >2
            JSR WSPARM      ;LENGTH =2, BYPASS
                             ABBREV SEARCH
            JMP NINE
GRT         JSR PREFIX      ;DO PREFIX CHECK
            LDA #40         ;SET ABBREV MODE
            ORA CNTRL4
            STA CNTRL4
            LDA #A7         ;INITIAL PARAMETER POINTERS
            STA PNTH
            STA WDTABH
            STA OFSETH
            LDA #DA
            STA PNTL
            LDA #ED
            STA WDTABL
            LDA #F5
            STA OFSETL
            LDA #03         ;SET MIN FILE LENGTH
            STA MINF
NINE        LDA #00
            STA FBNDX       ;INIT FILE BLOCK INDEX
            STA TEXTH       ;INIT TEXTH
            TAX
BKSR1       STA FB,X        ;CLEAR FILE BLOCK
            INX
            CPX #0F         ;SIXTEEN LOCATIONS
            BNE BKSR1
            LDA #60         ;SET FIRST FILE ENTRY
            STA 7E
            LDA CHRWD1      ;INIT TEXT FILE LENGTH
            STA 7F
SEVEN       LDX #FF         ;SEARCH FOR AN OPEN FILE
BKSR2       INX
            INX             ;GET 2'ND ENTRY
            LDY FB,X        ;B7=1 IMPLIES FILE CLOSED
            BMI BKSR2
            BNE OVRSR1      ;FILE CLOSED IF 00
EXITSR      RTS             ;TEXT WORD FINISHED
CVRSR1      STY CHRWD1      ;SET THE TEXT LENGTH
            LDA #80
            ORA FB,X        ;CLOSE LAST FILE
            STA FB,X
            LDY #09         ;SET FOR W.S MODE INITIALLY
```

```
                                ADJUST IF NOT
            BIT CNTRL4
            BVS W.S
            INY                 ;ABBREV
W.S         CPY CHRWD1          ;DETER SEARCH LENGTH
            BCC MAXY            ;SET Y TO MAX CHRWD1
            LDY CHRWD1          ;GET TEXT FILE LENGTH
MAXY        STY CHRWD2          ;SET CHRWD2 TO BE IN RANGE
            DEX                 ;GET TEXT BASE
            LDA FB,X            ;GET TEXTL
            STA TEXTL
            STA STXTL           ;SAVE BASE
EIGHT       SEC
            LDA CHRWD2          ;CALC W
            SBC CHRWD1
            STA W
TWO         SEC
            LDA CHRWD1          ;CALC CURRENT TEXT BASE
            SBC CHRWD2
            CLC                 ;SEARCH FROM RT TO LEFT
            ADC TEXTL
            STA TEXTL
FOUR        LDY CHRWD2          ;CALC TABLE PARAM FOR CHRWD2
            LDA (WDTAB),Y
            TAX                 ;GET NUMBER FILES THIS TABLE
            LDA (OFSET),Y       ;GET SPACING BETWEEN FILES
            STA OFF
            TYA
            ASL
            TAY
            LDA (PNT),Y         ;DERIVE TABLE POINTER
            STA TABLEL
ONE         LDY CHRWD2          ;GET FILE LENGTH
            DEY
THREE       LDA (TEXT),Y        ;GET TEXT CHAR
            CMP (TABLE),Y       ;COMPARE IT TO TABLE VALUE
            BEQ MATCH1          ;BRANCH IF MATCH
FIVE        DEX                 ;NOT A MATCH CONT
            BEQ SIX
            LDA TABLEL          ;MORE FILES THIS TABLE CALC
                                ;NEXT FILE THIS TABLE
            CLC
            ADC OFF
            STA TABLEL
            LDA TABLEH
            ADC #00
            STA TABLEH
            JMP ONE
MATCH1      CPY #00             ;SEE IF WHOLE WORD MATCH
            BEQ WMATCH
            DEY
            JMP THREE
WMATCH      BIT CNTRL4          ;COMPLETE MATCH
            BVS ABBREV
```

```
          LDY CHRWD2      ;DERIVE PNTR FOR W.S RULE
          LDA (TABLE),Y
          INY
          LDA (TABLE),Y
          CMP #FF         ;IF HIGH PART FF ACCEPT
          BEQ VLD
          STA RPNTL
          JMP (RPNT)      ;CHECK RULE
ABBREV    NOP
VLD       JSR CHGTXT
          JSR FBMAN
          JSR WSPARM
          JMP SEVEN       ;CONT IF NECESSARY
SIX       LDA W           ;NO MATCH THIS CHRWD2
          BNE ADJW
          LDY CHRWD2
          CPY MINF        ;REACHED MIN CHRWD2?
          BNE NEWY        ;IF NOT END THEN ADJUST Y
          BIT CNTRL4      ;YES CHECK MODE
          BVS ABRV        ;BRANCH FOR ABBREV
          JMP SEVEN
ABRV      JSR WSPARM
          JMP NINE        ;INIT FB THEN DO W.S SEARCH
ADJW      INC W           ;ADJUST W=W-1
          DEC TEXTL       ;CREATE NEW FILE IN TEXT SPACE
          JMP FOUR
NEWY      DEY             ;DECREASE SEARCH LENGTH
          STY CHRWD2
          JMP EIGHT       ;GO CALC NEW W AND NEW
                          ;PARAMETERS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WORDSIGN PARAMETER INITIALIZATION

```
PARAMETERS ARE SET TO THE WORDSIGN MODE
WSPARM    BIT CNTRL4
          BVC ALWAY       ;IF B6=0 BRANCH IN W.S MODE
          LDA #BF         ;SET FOR W.S MODE
          AND CNTRL4
          STA CNTRL4
          LDA #09         ;SET MAX POSSIBLE SEARCH LENGTH
          STA CHRWD2
          LDA CHRWD1
          CMP #09
          BCS OVRWS1
          STA CHRWD2      ;STORE CHRWD1 IN CHRWD2 ITS
                          ;LESS THAN 9
ALWAY     LDY #0D
          JSR INPNT
          RTS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

F. BRAILLE TRANSCRIPTION MACRO ROUTINES

SIMPLE UPPER WORDSIGNS

>

```
SUW'S   LDA CHRWD2
        CMP LENGTH
        BEQ OVRSU1
RJCTSU  JMP (INVLD)     ;REJECT IT
OVRSU1  LDA #20         ;CHECK FOR APOST
        AND CNTRL3
        BNE OVRSU2      ;BRANCH IF APOST
ACCEPT  JSR ADJC4       ;NO APOST
        JMP (VLD)       ;ACCEPT IT
OVRSU2  LDA #01         ;YES APOST
        AND CNTRL1      ;IS PCN SUC?
        BNE ACCEPT
        JMP RJCTSU
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WHOLE WORD CONTRACTIONS

```
A       LDA #01
        STA CHRWD2      ;SET FOR CHGTXT
        LDY #06         ;INDICATE AN "A"
        BNE WWC
AND     LDY #01
        BNE WWC
FOR     LDY #02
        BNE WWC
CF      LDY #03
        BNE WWC
THE     LDY #04
        BNE WWC
WITH    LDY #05
WWC     JSR CHEQ        ;L5 IS IMPLEMENTED HERE
        CMP #01         ;SEE IF SYLLABLE OR WHOLE  WORD
        BEQ WW          ;IF A WHOLE  WORD  BRANCH
        PHA
        JSR ADJC4       ;CLEAR FLAGS
        PLA             ;GET CODE
        CMP #02
        BEQ OK
        JMP (INVLD)
WW      LDA #10
        AND CNTRL1      ;SEE IF PCN PREC
        BEQ NOPCN
WWC3    TYA             ;SET WS CODE
        ORA CNTRL4
        STA CNTRL4
```

```
          CPY #06          ;ACCEPT WORD MATCH
          BNE OVRWW1
          RTS              ;THIS CAUSES RETURN TO PCN
OVRWW1    JMP (VLD)
NOPCN     LDA #07          ;GET CODE IF ANY OF PREVIOUS WS
          AND CNTRL4
          BNE PREVWS       ;IF CODE =00 BRANCH NO
                           ;PREVIOUS WS
          JMP WWC3
PREVWS    CPY #03          ;IS THIS WORD "OF"?
          BNE NXT1         ;BRANCH IF NOT AND
          JMP WWC3
NXT1      CPY #03          ;IS THIS WORD "OF"?
          BNE NXT2         ;CONT IF YES
          CMP #02          ;IS PREV WORD "FOR"?
          BNE WW4          ;BRANCH IF NOT "OF"
          JMP WWC3
NXT2      CPY #06          ;IS THIS WORD "A"
          BNE WWC4         ;IF NOT THEN BRANCH
          CMP #02          ;WAS PREV WORD "FOR"
          BNE WWC4         ;IF YES BRANCH
          JMP WWC3
WWC4      LDA #F8          ;CLEAR OLD CODE
          AND CNTRL4
          STA CNTRL4
          TYA              ;SET NEW CODE
          ORA CNTRL4
          STA CNTRL4
          LDY #00
          LDA #48          ;DELETE LAST SPACE IN BUFFER
          STA (SPNT),Y
OK        LDA CHRWD2
          CMP #01
          BNE OVRSU3
          RTS
OVRSU3    JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

UPPER CONTRACTIONS

```
UPCNCH    JSR SYLLAB       ;CHECK FOR OVERLAP OF
                            PREFIX/SUFFIX
          CMP #FF
          BEQ OVRUP1
          JMP (VLD)        ;ACCEPT IT
CVRUP1    JMP (INVLD)      ;REJECT IT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WHOLE WORD CONTRACTIONS

```
WWCON    LDA CHRWD2      ;IF NOT EXACT MATCH REJECT
         CMP LENGTH
         BNE OVRWWC
         JMP (VLD)
OVRWWC   JMP (INVLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

UPPER CONTRACTIONS, "ED, ER, CU, OW"

```
UPCNED   JSR DIPDIR      ;CHECK FOR DIP/DIR
         CMP #02
         BEQ CONTUP
BKUP1    CMP #FF
         BEQ RJCTUP
         JMP (VLD)       ;A=01 ACCEPT
RJCTUP   JMP (INVLD)
CONTUP   JSR SYLLAB      ;CHECK OVERLAP
         JMP BKUP1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

UPPER CONTRACTION, "ST

```
UPCONST  JSR SYLLAB      ;CHECK OVERLAP
         CMP #01
         BEQ OVRST1
RJCTST   JMP (INVLD)     ;REJECT
         LDY #01         ;SET Y TO START AREA OF TABLE
BKST     INY             ;BEGIN SEARCH FOR OWN
         CPY #05
         BEQ RJCTST      ;BRANCH IF TOTAL MATCH
         LDA (TEXT),Y    ;GET TEXT CHAR
         CMP TBLE,Y
         BEQ BKST
         JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
TBLE     OWN             ;THIS IS IN BRAILLE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

UPPER CONTRACTION, "AR"

```
UCNAR    TXA             ;SAVE X
         LDA #11         ;BRAILLE E
         TAX
         JSR DBLCHQ      ;SEE IF "EE"
         CMP #FF
         BNE OVRAR
         PLA             ;"EE" NOT FOUND
```

```
          TAX             ;NO TRIAGRAPH
CVRAR     PLA
          TAX
          JMP (INVLD)     ;REJECT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

UPPER CONTRACTIONS "ING, BLE"

```
UPCING    JSR SYLLAB      ;CHECK FOR OVERLAP
          CMP #01         ;BRANCH IF NO OVERLAP
          BEQ OVR
UPRJCT    JMP (INVLD)     ;EXIT AND REJECT
          LDA #0F
          AND PCNTRL      ;GET PREFIX LENGTH
          CLC             ;PREFIX IS NOT ALLOWED AT
                           START OF WORD
          ADC #60
          CMP TEXTL
          BEQ UPRJCT
          JMP (VLD)       ;ACCEPT IT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTION "COM"

```
LCNCOM    LDA #0F         ;ALLOW AT THE START OF THE
                          ;WORD ONLY
          AND TEXTL
          BEQ LCOVR
LCRJCT    JMP (INVLD)     ;EXIT
LCOVR     LDA #08         ;DASH NOT ALLOWED
          AND CNTRL2
          BNE LCRJCT
          JMP (VLD)       ;ACCEPT IT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTIONS "CON, DIS, BE"

```
LCNCON    LDA #0F         ;ALLOW ONLY AT BEGINNING
          AND TEXTL
          BNE CRJCT2
          TXA             ;SAVE X
          PHA
          LDA LENGTH      ;CALC ROOT LENGTH
          SEC
          SBC CHRWD2
          CMP #3
```

```
          BCS CCONT1      ;BRANCH IF ROOT >,=3
CRJCT3    PLA             ;RESTORE X
          TAX
CRJCT2    JMP (INVLD)     ;EXIT
CCONT1    JSR LCVWL       ;CHECK FOR A VOWEL IN ROOT
          CMP #01         ;A=01 IF VOWEL
          BNE CRJCT3      ;EXIT IF NO VOWEL
          LDA CHRWD2
          CMP #2
          BEQ LCBE        ;BRANCH IF=2
CCONT2    JSR LCONCQ      ;DO CHECK
          PLA
          TAX
CACCPT    JMP (VLD)       ;ACCEPT IT
LCBE      LDA 63          ;GET FOURTH CHAR
          CMP #11         ;MUST BE "E,A,I,O,U,H"
          BEQ FRCQ
CCONT3    JSR BESRCH      ;GO CHECK FOR SPECIAL SYMBOLS
          CMP #01
          BNE CRJCT3      ;IF YES EXIT
          JMP CCONT2
FRCQ      LDA 62
          CMP #N          ;RJCT VE,NE
          BEQ CRJCT3
          CMP #V
          BEQ RJCT 3
          JMP CCONT2      ;ACCEPT IT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTIONS "BB, CC, DD, FF, GG"

```
LCNBB     LDA #0F         ;CHECK FOR START OF WORD
          CMP TEXTL
          BEQ BBRJCT
          AND #0F         ;CHECK FOR END OF WORD
          CLC
          ADC CHRWD2
          CMP LENGTH
          BEQ BBRJCT
          JSR LCONCQ      ;DOT 1,4 CHECK
          JMP (VLD)
BBRJCT    JMP (INVLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER CONTRACTION "EN" AND WORDSIGN "ENOUGH"

```
EN        JSR LCONCQ      ;DOT 1,4
          JMP L71         ;L7 CONTAINS REST OF CHECKS
ENGH      LDA CHRWD2      ;MUST BE EXACT MATCH
          CMP LENGTH
```

```
          BEQ EXACT        ;DO NOT ALLOW DASH RULE
          JMP (INVLD)      ;EXIT
EXACT     LDA #3E          ;NO PCN ALLOWED
          AND CNTRL2
          BEQ ENOVR
ENOUT     JMP (INVLD)
ENOVR     LDA #FB          ;CHECK FOR PCN
          AND CNTRL3
          BNE ENOUT
          JMP (VLD)        ;ACCEPT IT
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER WORDSIGN "IN"

```
IN        LDA CHRWD2       ;EXACT MATCH ONLY
          CMP LENGTH
          BNE INOVR
INOUT     JMP EXACT
INOVR     JSR SYLLAB
          CMP #FF,
          BNE INCONT
          JMP (INVLD)
INCONT    JSR LCONCQ       ;CODE RETURNED IS IRRELAVENT
          JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

BE

```
BE        LDA CHRWD2       ;CHECK FOR EXACT MATCH
          CMP LENGTH
          BEQ INOUT
          JMP FORWRD
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

EA

```
EA        LDA #0F          ;SEE IF START
          AND TEXTL
          BEQ EARJCT
          CLC              ;CHECK END
          ADC CHRWD2
          CMP LENGTH
          BNE EACONT
EARJCT    JMP (INVLD)
EACONT    JSR CHGPNT       ;SWITCH PNTR
          LDA SCNTRL       ;SUFFIX CHECK
          AND #0F
          BEQ EATWO        ;IF NO SUFFIX CHECK PREFIX
```

```
                CMP #4          ;IF>=4 NO CHECK TO BE MADE
                BCS #4
                CMP #3
                BNE EANXT2
                LDY #2          ;IT IS LENGTH 3 CHECK "ATE"
                LDA (TEXT),Y
                CMP #1E         ;IS IT T?
                BNE EATWO
                DEY
                LDA (TEXT),Y
                CMP #11         ;IS IT E?
                BNE EATWO
EAONE           JSR CHGPNT      ;RESET TEXT PNTR
                JSR DIPDIR
                CMP #FF
                BEQ EARJC2
                DEC TEXTL
                LDY #00
                LDA (TEXT),Y
                INC TEXTL
                CMP #11         ;IS IT E?
                BEQ EARJC2
EACCPT          JSR LCONCQ      ;OK
                JMP (VLD)
EATWO           JSR SYLLABLE
                CMP #01
                BEQ EAONE
                JSR CHGPNT
EARJC2          JMP (INVLD)
EANXT2          LDY #02         ;CHECK FOR AL, AN
                LDA (TEXT),Y
                CMP #07
                BEQ EAONE
                CMP #1D
                BEQ EAONE
                JMP EATWO
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

LOWER WORDSIGNS "INTO, TO, BY"

```
L14             TXA
                PHA
                LDA CHRWD2
                CMP LENGTH      ;MUST BE EXACT MATCH
                BNE L14RJC
L14CNT          LDA #09         ;NO PCN OR CAPITALIZATION
                AND CNTRL1
                BNE L14RJC
                LDY #FF
                STY INDX14      ;INIT INDEX
L14BK           INC INDX14
                LDY INDX14      ;GET INDEX
```

```
          BMI L14RJC      ;IF NOTHING BUT SPACES REJECT
          LDA (CPNT),Y    ;GET CHAR
          JSR PCNCK       ;SEE WHAT IT IS
          CMP #00
          BEQ L14BK       ;IF NOT LETTER OR PCN BRANCH
          CMP #FF
          BEQ L14RJC
I14CNT    LDY INDX14      ;SET TEXT PNTR
          STY INDX2
          LDX #00         ;SPECIFIES FIRST WORD IN TABLE
          STX XINDX
          INX
          STX AINDX       ;SPECIFIES FIRST POINTER
                          ;IN TABLE
          JSR SRCHSET     ;CHECK FOR "AND, BUT, OR" AS
                          ;NEXT WORD FOLLOWING
          CMP #FF         ;IF FF NO MATCH
          BEQ DELSP
          LDA XINDX
          CMP #04
          BCS FLAG14
L14RJC    PLA
          TAX
          JMP (INVLD)
FLAG14    JSR CHAR        ;CURRENTLY PNTS TO SP
          CMP #20
          BNE L14OVR
          JSR DECPNT      ;ALLOW FOR ONLY ONE SP
          JMP L14RJC
          JSR DECPNT
DELSP     LDA #01
          CRA FLAG
          STA FLAG
          PLA
          TAX
          JMP (VLD)       ;SPACE ROUTINE WILL DELETE SPACE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

FINAL CONTRACTIONS

```
L20       LDA TEXTL       ;DO NOT ALLOW AT START OF WORD
          AND #0F
          BEQ L20RJC
          LDA #08
          AND CNTRL1      ;NO DASH
          BNE L20RJC
          JMP (VLD)
L20RJC    JMP (INVLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

COMPOUND SIGNS "EVER, HERE"

```
L15       LDA #0F
          AND TEXTL        ;ALLOW 2 BEFORE NONE AFTER
          BCC L15CNT
L15RJC    JMP (INVLD)
L15CNT    CLC
          ADC CHRWD2
          CMP LENGTH
          BNE L15RJC
          JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

INITIAL WORDSIGNS "TIME, SOME, PART, ONE"

```
TIME
L16       LDA #0F          ;2 BEFORE ANY AFTER
          AND TEXTL
          CMP #03
          BCS L16RJC
          JMP (VLD)

SOME      LDA CHRWD2       ;ALLOW ONLY EXACT MATCH
          CMP LENGTH
          BNE L16RJC
          JMP (VLD)

PART      LDY CHRWD2       ;H, AK NO TO FOLLOW
          LDA (TEXT),Y
          CMP #13
          BEQ L16RJC
          CMP #01
          BNE L16ACT
          INY
          LDA (TEXT),Y     ;GET NEXT CHAR
          CMP #05
          BEQ L16RJC
L16ACT    JMP (VLD)

L16ONE    LDA #0F          ;ONE BEFORE AND AFTER
                           ;BUT NOT D, N, R, TO FOLLOW
          AND TEXTL
          CMP #02
          BCS L16RJC
          CMP #01
          BEQ ONLTTR
          LDA LENGTH
          CMP #04          ;MUST BE AT START OF WORD
```

```
          BCC L16CNT
          BEQ L16CNT
L16RJC    JMP (INVLD)
ONLTTR    LDA LENGTH
          CMP #05          ;MUST BE ONE LETTER INFRONT
          BCC L16CNT       ;MUST BE LESS THAN OR EQUAL 5
          BEQ L16CNT
          JMP (INVLD)
L16CNT    LDY CHRWD2       ;GET NEXT CHAR
          LDA (TEXT),Y
          CMP #19          ;IS IT D?
          BEQ L16RJCT
          CMP #1D          ;IS IT N?
          BEQ L16RJC
          CMP #17          ;IS IT R?
          BEQ L16RJC
          JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

INITIAL WORDSIGNS "UNDER, WORD"

```
I16L18
UNDER     LDY CHRWD2       ;O, I IS NOT TO FOLLOW UNDER
          LDA (TEXT),Y
          CMP #0A          ;IS IT I?
          BEQ L16RJ2
          CMP #15          ;IS IT O?
          BEQ L16RJ2
          JMP (VLD)
L16RJ2    JMP (INVLD)

WORD      LDA #0F          ;NONE AFTER ANY AMOUNT BEFORE
          AND TEXTL
          BNE L16RJ2
          JMP (VLD)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

XII. REFERENCE TABLES

A. ABBREVIATION TABLES

++

10

++

```
.BYTE=$01,$0B,$1E,$11,$17,$2D,$01,$17,$19,$0E
                                        ;AFTERWARDS
.BYTE=$01,$0B,$3A,$8E,$00,$00,$00       ;AFWS


.BYTE=$01,$07,$1E,$15,$1B,$11,$1E,$13,$11,$17
                                        ;ALTOGETHER
.BYTE=$01,$07,$9E,$00,$00,$00,$00       ;ALT


.BYTE=$03,$11,$0B,$17,$0A,$11,$1D,$19,$11,$99
                                        ;BEFRIENDED
.BYTE=$06,$0B,$17,$0A,$22,$19,$AB       ;BEFRIENDED


.BYTE=$09,$15,$1D,$09,$11,$0A,$27,$0A,$1D,$1B
                                        ;CONCEIVING
.BYTE=$12,$09,$27,$9B,$00,$00,$00       ;CONCVG


.BYTE=$0F,$11,$17,$09,$11,$0A,$27,$0A,$1D,$9B
                                        ;PERCEIVING
.BYTE=$0F,$3B,$09,$27,$9B,$C0,$00       ;PERCVG


.BYTE=$1E,$13,$11,$0D,$0E,$11,$07,$27,$11,$0E
                                        ;THEMSELVES
.BYTE=$2E,$0D,$27,$8E,$00,$00,$00       ;THEMVS
```

```
.BYTE=$3D,$15,$25,$17,$0E,$11,$07,$27,$11,$0E
                                                ;YOURSELVES
.BYTE=$3D,$17,$27,$8E,$00,$00,$00               ;YRVS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

9

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$09,$09,$15,$17,$19,$0A,$1D,$1B       ;ACCORDING
.BYTE=$01,$89,$00,$00,$00,$00                   ;AC


.BYTE=$01,$0B,$1E,$11,$17,$1D,$15,$15,$1D       ;AFTERNOON
.BYTE=$01,$0B,$9D,$00,$00,$00                   ;AFN


.BYTE=$01,$0B,$1E,$11,$17,$3A,$01,$17,$19       ;AFTERWARD
.BYTE=$01,$0B,$BA,$00,$00,$00                   ;AFW


.BYTE=$19,$11,$09,$11,$0A,$27,$0A,$1D,$1B       ;DECEIVING
.BYTE=$19,$09,$27,$9B,$00,$00                   ;DCVG


.BYTE=$19,$11,$09,$07,$11,$17,$0A,$1D,$1B       ;DECLARING
.BYTE=$19,$09,$07,$9B,$00,$00                   ;DCLG


.BYTE=$0B,$17,$0A,$11,$1D,$19,$0A,$1D,$1B       ;FRIENDING
.BYTE=$0B,$17,$0A,$22,$19,$AC                   ;FRIENDING


.BYTE=$0A,$0D,$0D,$11,$19,$0A,$01,$1E,$11       ;IMMEDIATE
.BYTE=$0A,$0D,$8D,$00,$00,$00                   ;IMM


.BYTE=$ID,$11,$09,$11,$0A,$0E,$01,$17,$3D       ;NECESSARY
.BYTE=$1D,$11,$89,$00,$00,$00                   ;NEC


.BYTE=$17,$11,$09,$11,$0A,$27,$0A,$1D,$1B       ;RECEIVING
.BYTE=$17,$09,$27,$9B,$00,$00                   ;RCVG


.BYTE=$17,$11,$1A,$15,$0A,$09,$0A,$1D,$1B       ;REJOICING
.BYTE=$17,$1A,$1B,$9B,$00,$00                   ;RJCG
```

```
.BYTE=$15,$25,$17,$0E,$11,$17,$27,$11,$0E     ;OURSELVES
.BYTE=$33,$17,$27,$8E,$00,$00                 ;OURVS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

8

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$17,$1E,$13,$15,$25,$1B,$13         ;ALTHOUGH
.BYTE=$1,$07,$39,$00,$00,$00                  ;ALTH

.BYTE=$03,$07,$0A,$1D,$19,$0A,$1D,$1B         ;BLINDING
.BYTE=$03,$07,$15,$19,$AC,$00                 ;BLINDING

.BYTE=$09,$13,$0A,$07,$19,$17,$11,$1D         ;CHILDREN
.BYTE=$21,$9D,$00,$00,$00,$00                 ;CHN

.BYTE=$09,$15,$1D,$09,$11,$0A,$27,$11         ;CONCEIVE
.BYTE=$12,$09,$A7,$00,$00,$00                 ;CONCV

.BYTE=$0B,$17,$0A,$11,$1D,$19,$11,$19         ;FRIENDED
.BYTE=$0B,$17,$0A,$22,$19,$AB                 ;FRIENDED

.BYTE=$0F,$11,$17,$09,$11,$0A,$27,$11         ;PERCEIVE
.BYTE=$0F,$3B,$09,$A7,$00,$00                 ;PERCV

.BYTE=$1E,$15,$1B,$11,$1E,$13,$11,$17         ;TOGETHER
.BYTE=$1E,$1B,$97,$00,$00,$00                 ;TGR

.BYTE=$1E,$15,$0D,$0D,$15,$17,$15,$3A         ;TOMORROW
.BYTE=$1E,$8D,$00,$00,$00,$00                 ;TM

.BYTE=$3D,$15,$25,$17,$0E,$11,$07,$0B         ;YOURSELF
.BYTE=$3D,$17,$8D,$00,$00,$00                 ;YRF
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

7

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$1B,$01,$0A,$1D,$0E,$1E             ;AGAINST
.BYTE=$01,$1B,$8C,$00,$00                     ;AGST
```

```
.BYTE=$01,$07,$17,$11,$01,$19,$3D          ;ALREADY
.BYTE=$01,$07,$97,$00,$00                  ;ALR

.BYTE=$03,$11,$09,$01,$25,$0E,$11          ;BECAUSE
.BYTE=$06,$89,$00,$00,$00                  ;BEC

.BYTE=$03,$11,$1D,$11,$01,$1E,$13          ;BENEATH
.BYTE=$06,$9D,$00,$00,$00                  ;BEN

.BYTE=$03,$11,$1E,$3A,$11,$11,$1D          ;BETWEEN
.BYTE=$06,$9E,$00,$00,$00                  ;BET

.BYTE=$03,$07,$0A,$1D,$19,$11,$19          ;BLINDED
.BYTE=$03,$07,$14,$19,$2B                  ;BLINDED

.BYTE=$03,$17,$01,$0A,$07,$07,$11          ;BRAILLE
.BYTE=$03,$17,$87,$00,$00                  ;BRL

.BYTE=$19,$11,$09,$11,$0A,$27,$11          ;DECEIVE
.BYTE=$19,$09,$A7,$00,$00                  ;DCV

.BYTE=$19,$11,$09,$07,$11,$17,$11          ;DECLARE
.BYTE=$19,$09,$87,$00,$00                  ;DCL

.BYTE=$13,$11,$17,$0E,$11,$07,$0B          ;HERSELF
.BYTE=$07,$3B,$8B,$00,$00                  ;HERF

.BYTE=$13,$0A,$0D,$0E,$11,$07,$0B          ;HIMSELF
.BYTE=$13,$0D,$8B,$00,$00                  ;HMF

.BYTE=$1D,$11,$0A,$1E,$13,$11,$17          ;NEITHER
.BYTE=$1D,$11,$8A,$00,$00                  ;NEI

.BYTE=$15,$1D,$11,$0E,$11,$07,$0B          ;ONESELF
.BYTE=$10,$15,$8B,$00,$00                  ;ONEF

.BYTE=$0F,$11,$17,$13,$01,$0F,$0E          ;PERHAPS
.BYTE=$0F,$3B,$93,$00,$00                  ;PERH

.BYTE=$17,$11$09,$11,$0A,$27,$11           ;RECEIVE
```

```
.BYTE=$17,$09,$A7,$00,$00                    ;RCV

.BYTE=$17,$11,$1A,$15,$0A,$09,$11            ;REJOICE
.BYTE=$1,$1A,$89,$00,$00                     ;RJC

.BYTE=$1E,$13,$3D,$0E,$11,$07,$0B            ;THYSELF
.BYTE=$39,$3D,$8B,$00,$00                    ;THYF

.BYTE=$1E,$15,$1D,$0A,$1B,$13,$1E            ;TONIGHT
.BYTE=$1E,$9D,$00,$00,$00                    ;TN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

6

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$9,$17,$15,$0E,$0E                 ;ACROSS
.BYTE=$01,$09,$97,$00                        ;ACR

.BYTE=$01,$07,$0D,$15,$0E,$1E                ;ALMOST
.BYTE=$01,$07,$8D,$00                        ;ALM

.BYTE=$01,$07,$3A,$01,$27,$0E                ;ALWAYS
.BYTE=$01,$07,$BA,$00                        ;ALW

.BYTE=$03,$11,$0B,$15,$17,$11                ;BEFORE
.BYTE=$06,$8B,$00,$00                        ;BEF

.BYTE=$03,$11,$13,$0A,$1D,$19                ;BEHIND
.BYTE=$06,$93,$00,$00                        ;BEH

.BYTE=$03,$11,$0E,$0A,$1,$9,$11              ;BESIDE
.BYTE=$06,$8E,$00,$00                        ;BES

.BYTE=$03,$11,$3D,$15,$1D,$19                ;BEYOND
.BYTE=$06,$BD,$00,$00                        ;BEY

.BYTE=$11,$0A,$1E,$13,$11,$17                ;EITHER
.BYTE=$11,$8A,$00,$00                        ;EI

.BYTE=$0B,$17,$0A,$11,$1D,$19                ;FRIEND
.BYTE=$0B,$97,$00,$00                        ;FR
```

```
.BYTE=$0A,$1E,$0E,$11,$07,$0B                    ;ITSELF
.BYTE=$2D,$8B,$00,$00                            ;XF

.BYTE=$0D,$3D,$0E,$11,$07,$0B                    ;MYSELF
.BYTE=$0D,$3D,$0E,$8B                            ;MYSF

.BYTE=$0E,$13,$15,$25,$07,$19                    ;SHOULD
.BYTE=$29,$99,$00,$00                            ;SHD

.BYTE=$07,$11,$1E,$1E,$11,$17                    ;LETTER
.BYTE=$07,$97,$00,$00                            ;LR
.BYTE=$07,$0A,$1E,$1E,$07,$11                    ;LITTLE
.BYTE=$07,$87,$00,$00                            ;LL
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

5

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$03,$15,$25,$1E                        ;ABOUT
.BYTE=$01,$83,$00                                ;AB

.BYTE=$01,$03,$15,$27,$1                         ;ABOVE
.BYTE=$01,$03,$A7                                ;ABV

.BYTE=$01,$0B,$1E,$11,$17                        ;AFTER
.BYTE=$01,$8B,$00                                ;AF

.BYTE=$01,$1B,$01,$0A,$1D                        ;AGAIN
.BYTE=$01,$9B,$00                                ;AG

.BYTE=$03,$11,$07,$15,$3A                        ;BELOW
.BYTE=$06,$87,$00                                ;BEL

.BYTE=$03,$07,$0A,$1D,$19                        ;BLIND
.BYTE=$03,$87,$00                                ;BL

.BYTE=$09,$15,$25,$07,$19                        ;COULD
.BYTE=$09,$99,$00                                ;CD

.BYTE=$0B,$0A,$17,$0E,$1E                        ;FIRST
```

```
.BYTE=$0B,$8C,$00                          ;FST


.BYTE=$1B,$17,$11,$01,$1E                  ;GREAT
.BYTE=$1B,$17,$9E                          ;GRT


.BYTE=$1F,$25,$0A,$09,$05                  ;QUICK
.BYTE=$1F,$85,$00                          ;QK


.BYTE=$1E,$15,$19,$01,$3D                  ;TODAY
.BYTE=$1E,$99,$00                          ;TD


.BYTE=$3A,$15,$25,$07,$19                  ;WOULD
.BYTE=$3A,$99,$00                          ;WD
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$07,$0E,$15                      ;ALSO
.BYTE=$01$87                               ;AL


.BYTE=$1B,$15,$15,$19                      ;GOOD
.BYTE=$1E,$99                              ;GD


.BYTE=$0D,$25,$09,$13                      ;MUCH
.BYTE=$0D,$A1                              ;MCH


.BYTE=$0D,$25,$0E,$1E                      ;MUST
.BYTE=$0D,$8C                              ;MST


.BYTE=$0F,$01,$0A,$19                      ;PAID
.BYTE=$0F,$99                              ;PD


.BYTE=$0E,$01,$0A,$19                      ;SAID
.BYTE=$0E,$99                              ;SD


.BYTE=$0E,$25,$09,$13                      ;SUCH
.BYTE=$0E,$A1                              ;SCH


.BYTE=$3D,$15,$25,$17                      ;YOUR
.BYE=$3D,$97                               ;YR
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$13,$0A,$0D                                       ;HIM
.BYTE=$13,$8D                                           ;HM

.BYTE=$0A,$1E,$0E                                       ;ITS
.BYTE=$2D,$8E                                           ;XS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

B. WORDSIGN TABLE

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

9

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$09,$13,$01,$17,$01,$09,$1E,$11,$17     ;CHARACTER
.BYTE=$FF,$FF
.BYTE=$60,$A1

.BYTE=$05,$1D,$15,$3A,$07,$11,$19,$1B,$11     ;KNOWLEDGE
.BYTE=$BD,$38
.BYTE=$85,$00
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

8

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$1F,$25,$11,$0E,$1E,$0A,$15,$1D         ;QUESTION
.BYTE=$FF,$FF
.BYTE=$60,$BF
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

7

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$1E,$13,$17,$15,$25,$1B,$13             ;THROUGH
```

```
.BYTE=$FF,$FF
.BYTE=$60,$B9
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

6

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$09,$01,$1D,$1D,$15,$1E                    ;CANNOT
.BYTE=$FF,$FF
.BYTE=$78,$89


.BYTE=$11,$1D,$15,$25,$1B,$13                    ;ENOUGH
.BYTE=$C3,$60
.BYTE=$A2,$00


.BYTE=$0B,$01,$1E,$13,$11,$17                    ;FATHER
.BYTE=$FF,$FF
.BYTE=$60,$8B


.BYTE=$0F,$15,$1E,$13,$11,$17                    ;MOTHER
.BYTE=$FF,$FF
.BYTE=$60,$8D


.BYTE=$0F,$11,$15,$0F,$07,$11                    ;PEOPLE
.BYTE=$BD,$38
.BYTE=$8F,$00


.BYTE=$17,$01,$1E,$13,$11,$17                    ;RATHER
.BYTE=$BD,$38
.BYTE=$97,$00


.BYTE=$0E,$0F,$0A,$17,$0A,$1E                    ;SPIRIT
.BYTE=$FF,$FF
.BYTE=$78,$8E
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

5

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$1E,$0A,$15,$1D                        ;ATION
.BYTE=$C6,$50
.BYTE=$60,$9D
```

```
.BYTE=$09,$13,$0A,$07,$19                ;CHILD
.BYTE=$C4,$10
.BYTE=$A1,$00


.BYTE=$11,$27,$11,$17,$3D                ;EVERY
.BYTE=$BD,$38
.BYTE=$91,$00


.BYTE=$15,$25,$1B,$13,$1E                ;OUGHT
.BYTE=$FF,$FF
.BYTE=$50,$B3


.BYTE=$1F,$25,$0A,$1E,$11                ;QUITE
.BYTE=$FF,$FF
.BYTE=$9F,$00


.BYTE=$17,$0A,$1B,$13,$1E                ;RIGHT
.BYTE=$FF,$FF
.BYTE=$60,$97


.BYTE=$0E,$1E,$0A,$07,$07                ;STILL
.BYTE=$BD,$40
.BYTE=$8C,$00


.BYTE=$1E,$13,$11,$0A,$17                ;THEIR
.BYTE=$FF,$FF
.BYTE=$78,$AE


.BYTE=$1E,$13,$11,$17,$11                ;THERE
.BYTE=$FF,$FF
.BYTE=$50,$AE


.BYTE=$1E,$1B,$11,$0E,$11                ;THESE
.BYTE=$C6,$A0
.BYTE=$58,$2E


.BYTE=$1E,$13,$15,$0E,$11                ;THOSE
.BYTE=$FF,$FF
.BYTE=$58,$B9


.BYTE=$25,$1D,$19,$11,$17                ;UNDER
.BYTE=$C6,$90
.BYTE=$50,$A5
```

```
.BYTE=$3A,$13,$11,$17,$11                    ;WHERE
.BYTE=$FF,$FF
.BYTE=$50,$B1

.BYTE=$3A,$01,$0A,$09,$13                    ;WHICH
.BYTE=$C4,$10
.BYTE=$B1,$00

.BYTE=$3A,$13,$15,$0E,$11                    ;WHOSE
.BYTE=$FF,$FF
.BYTE=$58,$B1

.BYTE=$3A,$15,$17,$07,$19                    ;WORLD
.BYTE=$FF,$FF
.BYTE=$78,$BA

.BYTE=$0E,$13,$01,$07,$07                    ;SHALL
.BYTE=$C4,$10
.BYTE=$A9,$00

.BYTE=$3D,$15,$25,$1D,$1B                    ;YOUNG
.BYTE=$FF,$FF
.BYTE=$50,$BD
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$07,$07,$3D                        ;ALLY
.BYTE=$C6,$50
.BYTE=$60,$BD

.BYTE=$01,$1D,$09,$11                        ;ANCE
.BYTE=$C6,$50
.BYTE=$68,$91

.BYTE=$11,$1D,$09,$11                        ;ENCE
.BYTE=$C6,$50
.BYTE=$70,$91
.BYTE=$11,$27,$11,$17                        ;EVER
.BYTE=$C6,$70
.BYTE=$50,$91
```

```
.BYTE=$0B,$17,$15,$0D                ;FROM
.BYTE=$BD,$38
.BYTE=$8B,$00

.BYTE=$13,$01,$27,$11                ;HAVE
.BYTE=$BD,$38
.BYTE=$93,$00

.BYTE=$13,$11,$17,$11                ;HERE
.BYTE=$C6,$70
.BYTE=$50,$93

.BYTE=$0A,$1D,$1E,$15                ;INTO
.BYTE=$C6,$10
.BYTE=$54,$96

.BYTE=$1A,$25,$0E,$1E                ;JUST
.BYTE=$BD,$38
.BYTE=$9A,$00

.BYTE=$05,$1D,$15,$3A                ;KNOW
.BYTE=$FF,$FF
.BYTE=$50,$85

.BYTE=$07,$11,$0E,$0E                ;LESS
.BYTE=$C6,$50
.BYTE=$68,$8E

.BYTE=$07,$0A,$05,$11                ;LIKE
.BYTE=$BD,$38
.BYTE=$87,$00

.BYTE=$07,$15,$17,$09                ;LORD
.BYTE=$FF,$FF
.BYTE=$60,$87

.BYTE=$0D,$01,$1D,$3D                ;MANY
.BYTE=$FF,$FF
.BYTE=$78,$8D

.BYTE=$0D,$11,$1D,$1E                ;MENT
.BYTE=$C6,$50
.BYTE=$70,$BE
```

```
.BYTE=$0D,$15,$17,$11                    ;MORE
.BYTE=$BD,$38
.BYTE=$8D,$00


.BYTE=$1D,$01,$0D,$11                    ;NAME
.BYTE=$FF,$FF
.BYTE=$50,$9D


.BYTE=$1D,$11,$0E,$0E                    ;NESS
.BYTE=$C6,$50
.BYTE=$70,$8E


.BYTE=$15,$25,$1D,$19                    ;OUND
.BYTE=$C6,$50
.BYTE=$68,$99


.BYTE=$15,$25,$1D,$1E                    ;OUNT
.BYTE=$C6,$50
.BYTE=$68,$9E


.BYTE=$0F,$01,$17,$1E                    ;PART
.BYTE=$C6,$AC
.BYTE=$50,$8F


.BYTE=$0E,$0A,$15,$1D                    ;SION
.BYTE=$C6,$50
.BYTE=$68,$9D


.BYTE=$0E,$15,$0D,$11                    ;SOME
.BYTE=$C6,$90
.BYTE=$50,$8E


.BYTE=$1E,$13,$01,$1E                    ;THAT
.BYTE=$BD,$38
.BYTE=$9E,$00


.BYTE=$1E,$13,$0A,$0E                    ;THIS
.BYTE=$C4,$10
.BYTE=$B9,$00


.BYTE=$1E,$0A,$0D,$11                    ;TIME
.BYTE=$FF,$FF
```

```
.BYTE=$50,$9E

.BYTE=$1E,$0A,$15,$1D                    ;TION
.BYTE=$C6,$50
.BYTE=$70,$9D

.BYTE=$25,$0F,$15,$1D                    ;UPON
.BYTE=$C6,$A0
.BYTE=$4C,$A5

.BYTE=$2,$11,$17,$3D                     ;VERY
.BYTE=$BD,$38
.BYTE=$A7,$00

.BYTE=$3A,$1,$17,$11                     ;WERE
.BYTE=$C3,$60
.BYTE=$B6,$00

.BYTE=$3A,$0A,$07,$07                    ;WILL
.BYTE=$FF,$FF
.BYTE=$BA,$00

.BYTE=$3A,$0A,$1E,$13                    ;WITH
.BYTE=$B9,$7D
.BYTE=$BE,$00

.BYTE=$3A,$15,$17,$19                    ;WORD
.BYTE=$FF,$FF
.BYTE=$58,$BA

.BYTE=$3A,$15,$17,$05                    ;WORK
.BYTE=$FF,$FF
.BYTE=$50,$BA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$1D,$19                        ;AND
.BYTE=$B9,$6D
.BYTE=$AF,$00

.BYTE=$03,$07,$11                        ;BLE
```

```
.BYTE=$C0,$20
.BYTE=$BC,$00

.BYTE=$03,$25,$1E                    ;BUT
.BYTE=$BD,$38
.BYTE=$83,$00

.BYTE=$09,$11,$1D                    ;CAN
.BYTE=$BD,$38
.BYTE=$89,$00

.BYTE=$09,$15,$0D                    ;COM
.BYTE=$C1,$18
.BYTE=$A4,$00

.BYTE=$09,$15,$1D                    ;CON
.BYTE=$C1,$30
.BYTE=$92,$00

.BYTE=$19,$01,$3D                    ;DAY
.BYTE=$FF,$FF
.BYTE=$50,$99

.BYTE=$19,$0A,$0E                    ;DIS
.BYTE=$C1,$30
.BYTE=$B2,$00

.BYTE=$0B,$15,$17                    ;FOR
.BYTE=$B9,$71
.BYTE=$BF,$00

.BYTE=$0B,$25,$07                    ;FUL
.BYTE=$C6,$50
.BYTE=$70,$87

.BYTE=$13,$01,$19                    ;HAD
.BYTE=$C6,$A0
.BYTE=$78,$93

.BYTE=$13,$0A,$0E                    ;HIS
.BYTE=$C3,$60
.BYTE=$A6,$00
```

```
.BYTE=$0A,$1D,$1B                       ;ING
.BYTE=$C0,$20
.BYTE=$AC,$00


.BYTE=$0A,$1E,$3D                       ;ITY
.BYTE=$C6,$50
.BYTE=$70,$BD


.BYTE=$1D,$15,$1E                       ;NOT
.BYTE=$BD,$38
.BYTE=$9D,$00


.BYTE=$15,$1D,$11                       ;ONE
.BYTE=$C6,$CA
.BYTE=$50,$95


.BYTE=$15,$1D,$1B                       ;ONG
.BYTE=$C6,$50
.BYTE=$70,$9B


.BYTE=$15,$25,$1E                       ;OUT
.BYTE=$BD,$38
.BYTE=$B3,$00


.BYTE=$1E,$13,$1                        ;THE
.BYTE=$B9,$79
.BYTE=$AE,$00


.BYTE=$3A,$01,$0E                       ;WAS
.BYTE=$C3,$60
.BYTE=$B4,$00


.BYTE=$3D,$15,$25                       ;YOU
.BYTE=$BD,$38
.BYTE=$BD,$00
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

2

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$17                           ;AR
.BYTE=$C0,$00
.BYTE=$9C
```

```
.BYTE=$01,$0E                          ;AS
.BYTE=$BD,$38
.BYTE=$B5


.BYTE=$03,$03                          ;BB
.BYTE=$C3,$10
.BYTE=$86


.BYTE=$03,$11                          ;BE
.BYTE=$C1,$30
.BYTE=$86


.BYTE=$03,$3D                          ;BY
.BYTE=$C6,$10
.BYTE=$B4


.BYTE=$09,$09                          ;CC
.BYTE=$C3,$10
.BYTE=$92


.BYTE=$09,$13                          ;CH
.BYTE=$C4,$00
.BYTE=$A1


.BYTE=$19,$19                          ;DD
.BYTE=$C3,$10
.BYTE=$B2


.BYTE=$19,$15                          ;DO
.BYTE=$BD,$38
.BYTE=$99


.BYTE=$11,$01                          ;EA
.BYTE=$C3,$50
.BYTE=$82


.BYTE=$11,$19                          ;ED
.BYTE=$BE,$80
.BYTE=$AB


.BYTE=$11,$1D                          ;EN
.BYTE=$C3,$46
```

```
.BYTE=$A2

.BYTE=$11,$17                          ;ER
.BYTE=$BE,$80
.BYTE=$BB

.BYTE=$0B,$0B                          ;FF
.BYTE=$C3,$10
.BYTE=$96

.BYTE=$1B,$1B                          ;GG
.BYTE=$C3,$10
.BYTE=$B6

.BYTE=$1B,$13                          ;GH
.BYTE=$C4,$00
.BYTE=$A3

.BYTE=$1B,$15                          ;GO
.BYTE=$BD,$38
.BYTE=$9B

.BYTE=$0A,$1D                          ;IN
.BYTE=$C3,$85
.BYTE=$94

.BYTE=$0A,$1E                          ;IT
.BYTE=$FF,$FF
.BYTE=$AD

.BYTE=$15,$0B                          ;OF
.BYTE=$B9,$75
.BYTE=$B7

.BYTE=$15,$25                          ;OU
.BYTE=$BE,$95
.BYTE=$B3

.BYTE=$15,$3A                          ;OW
.BYTE=$BE,$95
.BYTE=$AA

.BYTE=$0E,$13                          ;SH
```

```
.BYTE=$C4,$00
.BYTE=$A9


.BYTE=$0E,$15                                   ;SO
.BYTE=$FD,$38
.BYTE=$8E


.BYTE=$0E,$1E                                   ;ST
.BYTE=$BE,$A8
.BYTE=$8C

.
.BYTE=$1E,$13                                   ;TH
.BYTE=$C4,$00
.BYTE=$B9


.BYTE=$1E,$15                                   ;TO
.BYTE=$C6,$10
.BYTE=$96


.BYTE=$25,$0E                                   ;US
.BYTE=$BD,$38
.BYTE=$A5


.BYTE=$3A,$13                                   ;WH
.BYTE=$B1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

C. ALPHABET AND NUMBER TABLES

```
.BYTE=$61,$01                                   ;A,1
.BYTE=$62,$03                                   ;B,2
.BYTE=$63,$09                                   ;C,3
.BYTE=64,$19                                    ;D,4
.BYTE=$65,$11                                   ;E,5
.BYTE=$66,$0B                                   ;F,6
.BYTE=$67,$1B                                   ;G,7
.BYTE=$68,$13                                   ;H,8
.BYTE=$69,$0A                                   ;I,9
.BYTE=$6A,$1A                                   ;J,0
.BYTE=$6B,$05                                   ;K
.BYTE=$6C,$07                                   ;L
.BYTE=$6D,$0D                                   ;M
.BYTE=$6E,$1D                                   ;N
.BYTE=$6F,$15                                   ;O
```

```
.BYTE=$70,$0F                                   ;P
.BYTE=$71,$1F                                   ;Q
.BYTE=$72,$17                                   ;R
.BYTE=$73,$0E                                   ;S
.BYTE=$74,$1E                                   ;T
.BYTE=$75,$25                                   ;U
.BYTE=$76,$27                                   ;V
.BYTE=$77,$3A                                   ;W
.BYTE=$78,$2D                                   ;X
.BYTE=$79,$3D                                   ;Y
.BYTE=$7A,$35                                   ;Z
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

D. PREFIX TABLES

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$09,$0A,$17,$09,$25,$0D                   ;CIRCUM

.BYTE=$09,$15,$1D,$1E,$17,$11                   ;CONTRA

.BYTE=$09,$15,$1D,1E,$17,$11                    ;CONTRE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

5

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$0A,$1D,$0B,$17,$01                       ;INFRA

.BYTE=$11,$2D,$1E,$17,$01                       ;EXTRA

.BYTE=$11,$2D,$1E,$17,$15                       ;EXTRO

.BYTE=0D,$25,$07,$1E,$0A                        ;MULTI

.BYTE=$13,$3D,$0F,$11,$17                       ;HYPER

.BYTE=$13,$D,$19,$17,$15                        ;HYDRO

.BYTE=$0E,$25,$0F,$11,$17                       ;SUPER
```

```
.BYTE=$1E,$17,$01,$1D,$0E                    ;TRANS

.BYTE=$17,$11,$1E,$17,$15                    ;RETRO

.BYTE=$0A,$1D,$1E,$11,$17                    ;INTER
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$0D,$03,$0A                        ;AMBI

.BYTE=$01,$1D,$1E,$0A                        ;ANTI

.BYTE=$01,$1D,$1E,$11                        ;ANTE

.BYTE=$0F,$15,$0E,$1E                        ;POST

.BYTE=$0,$11,$0D,$0A                         ;SEMI

.BYTE=$27,$0A,$09,$11                        ;VICE

.BYTE=$0F,$17,$01,$11                        ;PRAE

.BYTE=$0D,$15,$1D,$15                        ;MONO

.BYTE=$15,$27,$11,$17                        ;OVER

.BYTE=$0F,$01,$17,$01                        ;PARA

.BYTE=$0F,$15,$07,$3D                        ;POLY

.BYTE=$0F,$11,$17,$0A                        ;PERI

.BYTE=$1E,$11,$07,$11                        ;TELE
```

```
.BYTE=$09,$01,$1E,$01                                   ;CATA

.BYTE=$13,$15,$0D,$15                                   ;HOMO

.BYTE=$13,$D,$0F,$15                                    ;HYPO
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$09,$25,$0D                                       ;CUM

.BYTE=$09,$15,$0D                                       ;COM

.BYTE=$09,$15,$1D                                       ;CON

.BYTE=$09,$15,$17                                       ;COR

.BYTE=$19,$0A,$01                                       ;DIA

.BYTE=$19,$0A,$0E                                       ;DIS

.BYTE=$19,$0A,$0B                                       ;DIF

.BYTE=$11,$0F,$0A                                       ;EPI

.BYTE=$1D,$15,$1D                                       ;NON

.BYTE=$0F,$11,$17                                       ;PER

.BYTE=$0F,$17,$15                                       ;PRO

.BYTE=$0E,$25,$03                                       ;SUB

.BYTE=$0E,$25,$09                                       ;SUC
```

```
.BYTE=$0E,$25,$0B                                   ;SUF

.BYTE=$0E,$25,$1B                                   ;SUG

.BYTE=$0E,$25,$0D                                   ;SUM

.BYTE=$0D,$01,$07                                   ;MAL

.BYTE=$0F,$17,$11                                   ;PRE

.BYTE=$1E,$17,$0A                                   ;TRI

.BYTE=$25,$1D,$0A                                   ;UNI

.BYTE=$01,$1D,$01                                   ;ANA

.BYTE=$01,$0F,$15                                   ;APO

.BYTE=$09,$15,$07                                   ;COL

.BYTE=$0E,$25,$0E                                   ;SUS

.BYTE=$0E,$3D,$1D                                   ;SYN

.BYTE=$0E,$D ,$0D                                   ;SYM

.BYTE =$0E,$3D,$0E                                  ;SYS

.BYTE=$0D,$0A,$0E                                   ;MIS

.BYTE=$0E,$11,$3D                                   ;SED
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```


++

```
.BYTE=$1,$03                    ;AB
.BYTE=$01,$19                   ;AD
.BYTE=$03,$0A                   ;BI
.BYTE=$09,$15                   ;CO
.BYTE=$19,$11                   ;DE
.BYTE=$19,$0A                   ;DI
.BYTE=$11,$1D                   ;EN
.BYTE=$11,$09                   ;EC
.BYTE=$11,$0B                   ;EF
.BYTE=$0A,$1D                   ;IN
.BYTE=$0A,$07                   ;IL
.BYTE=$0A,$0D                   ;IM
.BYTE=$0A,$17                   ;IR
.BYTE=$15,$03                   ;OB
.BYTE=$15,$09                   ;OC
.BYTE=$15,$07                   ;OF
.BYTE=$15,$0F                   ;OP
.BYTE=$17,$11                   ;RE
```

```
.BYTE=$0E,$11                                   ;SE

.BYTE=$01,$09                                   ;AC

.BYTE=$01,$1B                                   ;AG

.BYTE=$01,$07                                   ;AL

.BYTE=$01,$1D                                   ;AN

.BYTE=$01,$0F                                   ;AP

.BYTE=$01,$17                                   ;AR

.BYTE=$01,$0E                                   ;AS

.BYTE=$01,$1E                                   ;AT

.BYTE=$03,$11                                   ;BE

.BYTE=$11,$0D                                   ;EM
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

E. SUFFIX TABLES

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

6

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$01,$1E,$17,$11,$0E,$0E                   ;STRESS
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

5

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$11,$0E,$1F,$25,$11                                ;ESQUE

.BYTE=$01,$07,$15,$1B,$3D                                ;ALOGY
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$09,$25,$07,$11                                    ;CULE

.BYTE=$11,$1E,$1E,$11                                    ;ETTE

.BYTE=$07,$0A,$1D,$1B                                    ;LING

.BYTE=$01,$03,$07,$11                                    ;ABLE

.BYTE=$0A,$03,$07,$11                                    ;IBLE

.BYTE=$01,$1D,$09,$11                                    ;ANCE

.BYTE=$11,$1D,$09,$11                                    ;ENCE

.BYTE=$1E,$25,$19,$11                                    ;TUDE

.BYTE=$0B,$25,$07,$07                                    ;FULL

.BYTE=$07,$11,$0E,$0E                                    ;LESS

.BYTE=$1E,$0A,$15,$1D                                    ;TION

.BYTE=$0D,$11,$1D,$1E                                    ;MENT

.BYTE=$0E,$13,$0A,$0F                                    ;SHIP

.BYTE=$07,$0A,$05,$11                                    ;LIKE
```

```
.BYTE=$11,$0E,$09,$11                    ;ESCE

.BYTE=$13,$15,$15,$19                    ;HOOD

.BYTE=$1D,$11,$0E,$0E                    ;NESS

.BYTE=$3A,$01,$17,$19                    ;WARD

.BYTE=$07,$15,$1B,$3D                    ;LOGY
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```



```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$07,$11,$1E                        ;LET

.BYTE=$15,$09,$05                        ;OCK

.BYTE=$25,$07,$11                        ;ULE

.BYTE=$05,$0A,$1D                        ;KIN

.BYTE=$11,$0E,$0E                        ;ESS

.BYTE=$0A,$1E,$3D                        ;ITY

.BYTE=$01,$1D,$1E                        ;ANT

.BYTE=$11,$1D,$1E                        ;ENT

.BYTE=$0A,$15,$1D                        ;ION

.BYTE=$0A,$01,$1D                        ;IAN

.BYTE=$0A,$03,$13                        ;ISH
```

```
.BYTE=$15,$25,$0E                       ;OUS

.BYTE=$01,$1B,$11                       ;AGE

.BYTE=$01,$17,$19                       ;ARD

.BYTE=$01,$1E,$11                       ;ATE

.BYTE=$19,$15,$0D                       ;DOM

.BYTE=$11,$11,$17                       ;EER

.BYTE=$11,$17,$3D                       ;ERY

.BYTE=$0A,$09,$11                       ;ICE

.BYTE=$0A,$17,$11                       ;ILE

.BYTE=$0A,$E,$0D                        ;ISM
.BYE=$0A,$0E,$1E                       ;IST

.BYTE=$0A,$27,$11                       ;IVE

.BYTE=$0A,$35,$11                       ;IZ

.BYTE=$15,$17,$3D                       ;ORY

.BYTE=$15,$0E,$11                       ;OSE

.BYTE=$25,$17,$11                       ;URE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

2

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.BYTE=$07,$3D                           ;LY
```

```
.BYTE=$11,$17                    ;ER
.BYTE=$15,$17                    ;OR
.BYTE=$11,$07                    ;EL
.BYTE=$11,$1E                    ;ET
.BYTE=$0A,$1D                    ;IN
.BYTE=$07,$11                    ;LE
.BYTE=$01,$09                    ;AC
.BYTE=$01,$07                    ;AL
.BYTE=$09,$3D                    ;CY
.BYTE=$11,$11                    ;EE
.BYTE=$11,$1D                    ;EN
.BYTE=$11,$17                    ;ER
.BYTE=$0B,$3D                    ;FY
.BYTE=$07,$3D                    ;LY
.BYTE=$15,$17                    ;OR
.BYTE=$1E,$13                    ;TH
.BYTE=$1E,$3D                    ;TY
.BYTE=$01,$17                    ;AR
```

```
.BYTE=$0A,$09                                          ;IC
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

F. INDEXED TABLES

PREADD

```
.BYTE=$39                                              ;1

.BYTE=$FB                                              ;2

.BYTE=$92                                              ;3

.BYTE=$4A                                              ;4

.BYTE=$1D                                              ;5

.BYTE=$00                                              ;6
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PRETAB

```
.BYTE=$02                                              ;1

.BYTE=$1D                                              ;2

.BYTE=$21                                              ;3

.BYTE=$10                                              ;4

.BYTE=$09                                              ;5

.BYTE=$04                                              ;6
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SUFADD

```
.BYTE=$08                                   ;2
.BYTE=$AE                                   ;3
.BYTE=$56                                   ;4
.BYTE=$4C                                   ;5
.BYTE=$40                                   ;6
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

SUFTAB

```
.BYTE=$15                                   ;2
.BYTE=$1C                                   ;3
.BYTE=$14                                   ;4
.BYTE=$02                                   ;5
.BYTE=$02                                   ;6
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WSPNT

```
.BYTE=$A6,$CE                               ;2
.BYTE=$A6,$3B                               ;3
.BYTE=$A5,$23                               ;4
.BYTE=$A4,$81                               ;5
.BYTE=$A4,$31                               ;6
```

```
.BYTE=$A4,$26                                   ;7

.BYTE=$A4,$1A                                   ;8

.BYTE=$A4,$00                                   ;9
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WRDTBW

```
.BYTE=$1D                                       ;2

.BYTE=$15                                       ;3

.BYTE=$22                                       ;4

.BYTE=$12                                       ;5

.BYTE=$07                                       ;6

.BYTE=$01                                       ;7

.BYTE=$01                                       ;8

.BYTE=$02                                       ;9
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WOFFST

```
.BYTE=$05                                       ;2

.BYTE=07                                        ;3

.BYTE=$08                                       ;4

.BYTE=$09                                       ;5
```

```
.BYTE=$0A                                                    ;6

.BYTE=$0B                                                    ;7

.BYTE=$0C                                                    ;8

.BYTE=$0D                                                    ;9
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ABVPNT

```
.BYTE=$A3,$E0                                                ;3

.BYTE=$A3,$AA                                                ;4

.BYTE=$A3,$42                                                ;5

.BTE=$A3,$AC                                                 ;6

.BYTE=$A1,$C8                                                ;7

.BYTE=$A1,$3C                                                ;8

.BYTE=$A0,$88                                                ;9

.BYTE=$A0,$00                                                ;A
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

WDTBAB

```
.BYTE=$02                                                    ;3

.BYTE=$08                                                    ;4

.BYTE=$0C                                                    ;5
```

```
.BYTE=$0E                                                    ;6

.BYTE=$12                                                    ;7

.BYTE=$09                                                    ;8

.BYTE=$0B                                                    ;9

.BYTE=$07                                                    ;A
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ABOFST

```
.BYTE=$05                                                    ;3

.BYTE=$06                                                    ;4

.BYTE=$08                                                    ;5

.BYTE=$0A                                                    ;6

.BYTE=$0C                                                    ;7

.BYTE=$0E                                                    ;8

.BYTE=$0F                                                    ;9

.BYTE=$11                                                    ;A
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```